Konzeption und Umsetzung eines Werkzeugs zur Definition von Navigationsflüssen mittels Dienstannotationen

Diplomarbeit

am Institut für Systemarchitektur

vorgelegt von: Felix Martens

Studiengang: Medieninformatik

Hochschullehrer: Prof. Dr. rer. nat. habil. Dr. h. c. Alexander Schill

Betreuer: Dipl.-Inf. Marius Feldmann

2010

Selbständigkeitserklärung

Ich versichere hiermit, dass ich meine Diplomarbeit mit dem Thema

> *Konzeption und Umsetzung eines Werkzeugs zur Definition von Navigationsflüssen mittels Dienstannotationen*

selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Dresden, den 30.08.2010

Felix Martens

Inhaltsverzeichnis

1. Einleitung

1.1. Motivation

Die zunehmend lose Kopplung von Softwarekomponenten innerhalb von IT-Infrastrukturen war ein Trend der letzten Jahre. Laut Gartner's Hype-Zyklus-Analyse[Gar09] entwickeln sich Serviceorientierte Architekturen (SOA) nach dem Hype der ersten Jahre und einer Phase der Ernüchterung mittlerweile zu einem etablierten Architekturmuster der Softwareentwicklung und finden bei immer mehr IT-Firmen Akzeptanz. Basiselemente der Serviceorientierten Architekturen sind Dienste, die über funktionale Schnittstellen Informationen zum Zugriff auf Dienstfunktionalitäten zur Verfügung stellen. Die Schnittstellenbeschreibungen wurden aber ausschließlich unter dem Aspekt der Maschine-zu-Maschine Kommunikation entwickelt, um einen standardisierten Datenaustausch zwischen verteilten Systemen zu ermöglichen. Ein direkter Zugriff auf Dienstfunktionalitäten durch einen Nutzer wurde aufgrund dieser Fokussierung nicht berücksichtigt. Daher müssen die fehlenden Benutzerschnittstelleninformationen der Dienste durch eine manuelle Entwicklung von Client-Softwarekomponenten ausgeglichen werden. Da diese statische Form der Benutzerschnittstellenerzeugung sehr aufwendig ist und im Fall einer Änderung der funktionalen Schnittstellenbeschreibung manuelle Anpassungen notwendig sind, wurden Inferenzmechanismen entwickelt, die automatisiert Eingabemasken erstellen und über diese einen direkten Dienstaufruf ermöglichen. Die dazu notwendigen Informationen werden aus den Schnittstellenbeschreibungen der Diensten abgeleitet, indem für die Eingabeparameter von Operationen Eingabefelder generiert werden und die Rückgabe eines Dienstaufrufes auf einer separaten Seite dargestellt wird. Die Qualität dieser generierten Eingabemasken ist allerdings aufgrund der Bindung an die Bezeichnungen der funktionalen Schnittstellenbeschreibung stark eingeschränkt. Daher wurde die funktionale Beschreibung von Diensten in weiteren Ansätzen durch zusätzliche Benutzerschnittstellenbeschreibungen, welche im Folgenden als *Dienstannotationen* bezeichnet werden, erweitert. Dienstannotationen können in den Generierungsprozess eingebunden werden und die Qualität der generierten Benutzerschnittstellen deutlich verbessern. Obwohl die Generierung von Benutzerschnittstellen für Dienste in den letzten Jahren einen bedeutenden Gegenstand der Forschung darstellte und Lösungsansätze wie beispielsweise die Inferenzengine Dynvoker [Spi06a] hervorbrachte, sind die erzielten Ergebnisse weiter verbesserungswürdig. Die Ausdrucksschwäche verwendeter Annotationsmodelle und der vordefinierte Navigationsfluss generierter Benutzerschnittstellen, welcher die Navigation auf den Aufruf einer Eingabemaske für eine Dienstoperation und das Anzeigen einer Rückgabe beschränkt, limitieren das Potential dieser Verfahren.

1.2. Einordnung in den Gesamtansatz

Der neue Ansatz, welcher im Rahmen der Dissertation [Fel11] entstanden ist, greift das Prinzip der Benutzerschnittstellenbschreibung mit Hilfe annotierter Dienstschnittstellenbeschreibungen auf. Neben Dienstannotationen, die das Verhalten einzelner Benutzerschnittstellenelemente beschreiben, sollen zusätzliche Dienstannotationen zur Abbildung von Navigations- und Datenflüssen eingeführt werden. Das Ziel des Ansatzes besteht darin, dienstbasierte, interaktive Anwendungen vollständig beschreiben zu können und zur Ausführung auf verschiedenen Plattformen verfügbar zu machen. Abbildung 1.1 gibt einen Überblick über das Gesamtsystem.

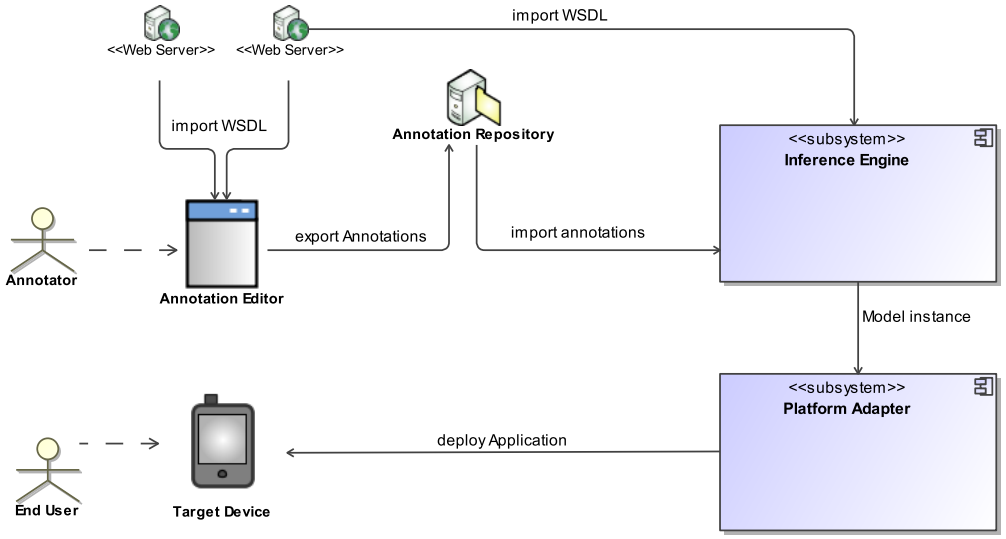


Abbildung 1.1.: Überblick über das Gesamtsystem

Diese Arbeit beschäftigt sich mit der Konzeption und Umsetzung eines *Annotationeneditors*, welcher den ersten Teil des Gesamtsystems ausmacht. Ein Annotator soll mit Hilfe des Editors die Möglichkeit haben interaktive Anwendungen zu modellieren. Dabei können beliebig viele Dienste in die Modellierung einer Anwendung einbezogen werden. Nach der Fertigstellung werden die Anwendungsbeschreibungen in Form von Dienstannotationen in einem speziellen Verzeichnisdienst (Repository) veröffentlicht und stehen dort der *Inferenzengine*, die in [Fel11] beschrieben wird, zur Weiterverarbeitung zur Verfügung. Die Inferenzengine erzeugt aus den Dienstannotationen einer Anwendungsbeschreibung und den Informationen der funktionalen Schnittstellenbeschreibungen integrierter Dienste Instanzen eines Modells zur Beschreibung interaktiver Komponenten. Eine interaktive Komponente ist eine interaktive, dienstbasierte Anwendung durch die Bereitstellung einer Benutzungsschnittstelle. Mit Hilfe unterschiedlicher Zielspezifikationen, die plattformspezifische Besonderheiten zur Integration abbilden, kann die Inferenzengine aus einer Anwendungsbeschreibung automatisch angepasste Instanzen für unterschiedliche Plattformen generieren. Um eine interaktive Komponente letztendlich auf einer spezifischen Plattform ausführen zu können, muss diese von einem *Platform Adapter* in eine auf dieser Plattform ausführbaren Komponente transformiert und veröffentlicht werden. Über eine Zielanwendung kann

die resultierende Anwendung dann auf das Zielgerät heruntergeladen und in Betrieb genommen werden. Dieser letzte Teil der Kette wird parallel zu dieser Arbeit mittels einer Modell-zu-Code Transformation für die Android-Plattform umgesetzt [Ber10].

1.3. Ziele der Arbeit

Das Ziel dieser Arbeit besteht darin, ein Autorenwerkzeug zur Modellierung von interaktiven, dienstbasierten Anwendungen zu konzipieren und umzusetzen. Das Werkzeug soll nahtlos in die Prozesskette, welche im vorherigen Absatz beschrieben wurde, integriert werden können. Als gemeinsame Schnittstelle soll hierzu im Rahmen dieser Arbeit ein Format zur Beschreibung anwendungsrelevanter Funktionalitäten in Form von Dienstannotationen erstellt werden. Mit Hilfe der Dienstannotationen sollen Navigations- und Datenflüsse, sowie weitere wiederkehrende Muster und Eigenschaften von Benutzeroberflächen beschreibbar sein.

Aus den Zielen dieser Arbeit lassen sich für die Umsetzung folgende zentrale Fragestellungen formulieren:

- **Wie können Dienstannotationen zum Beschreiben von Navigations- und Datenflüsse, sowie weiterer wiederkehrender Bedienungsmuster einer Anwendung, ausgedrückt werden?**
- **Wie kann die Nutzerinteraktion in einem graphischen Editor gestaltet werden, um eine benutzerfreundliche Modellierung von interaktiven, dienstbasierten Anwendungen zu ermöglichen?**
- **Welche Beschreibungsmodelle werden zur Umsetzung des Editors benötigt und wie muss die Softwarearchitektur des Editors aufgebaut sein, um die Integration und interne Kommunikation der Modelle zu realisieren?**

1.4. Überblick über die Arbeit

Die Arbeit gliedert sich in sieben Kapitel, die stark an den Entwicklungsprozess der Softwarekomponente angelehnt sind. Kapitel 2 beschäftigt sich zunächst mit den Grundlagen aus dem Umfeld der automatisierten Benutzerschnittstellengenerierung für Dienste. Nach einer allgemeinen Einführung etablierter Lösungsansätze werden aktuelle Verfahren und Technologien in einer Stand-der-Technik-Analyse detailliert untersucht und verglichen. Die aus dieser Untersuchung gewonnenen Erkenntnisse fließen in die Analyse (Kapitel 3) des neuen Ansatzes ein, indem in einem ersten Schritt ein geeignetes Anwendungsszenario aufgestellt wird und daraus dann Anforderungen an das zu realisierende Softwarewerkzeug und an die zu konzipierenden Dienstannotationen abgeleitet werden. In Kapitel 4 wird durch die Konzeption des Ansatzes eine theoretische Grundlage für die Umsetzung geschaffen. Vor der Architekturbeschreibung eines graphischen Editors zur Modellierung von interaktiven,

dienstbasierten Anwendungen, werden in diesem Kapitel zunächst Dienstannotationen als Grundlage für die Abbildung von Informationen zur Anwendungsbeschreibung entworfen. Kapitel 5 beschreibt eine konkrete Umsetzung der Softwarekomponente als Prototyp, indem auf die Umsetzung einzelner Systemkomponenten eingegangen wird und bedeutende Aspekte bei der Implementierung hervorgehoben werden. Zur Bewertung des Prototypen werden in Kapitel 6 die aufgestellten Anforderungen anhand einer vorangegangenen Szenarienvalidierung evaluiert. Aufgrund der besonderen Bedeutung der Nutzbarkeit des realisierten Softwarewerkzeugs, wird diese mit Hilfe eines Usability-Tests evaluiert. Abgeschlossen wird das Kapitel mit der Evaluierung des im vorangegangenen Abschnitt vorgestellten Gesamtkonzepts. Im letzten Kapitel werden die grundlegenden Inhalte und Ergebnisse der Arbeit resümiert und in einem abschließenden Ausblick mögliche Erweiterungen beziehungsweise offene Fragen herausgestellt.

2. Grundlagen

Web Services wurden vor allem unter dem Aspekt der Maschine-zu-Maschine Kommunikation entwickelt. Durch Sie soll eine standardisierte Kommunikation zwischen Anwendungen ermöglicht werden. Die direkte Kommunikation zwischen einem Benutzer und einem Dienst wurde aber aufgrund dieser Fokussierung auf die Anwendungskommunikation nicht berücksichtigt.

Im Folgenden werden Methoden vorgestellt, die den Mangel der Benutzer-zu-Maschine Kommunikation umgehen, indem Benutzeroberflächen für Dienste generiert werden. Während im Abschnitt 2.1 zunächst auf die Generierung von Benutzerschnittstellen alleine aus den Informationen der Dienstbeschreibungen eingegangen wird, zeigen die darauf folgenden Abschnitte Möglichkeiten auf, das Generierungsresultat durch Einbeziehung zusätzlicher Informationen zu verbessern. Im Abschnitt 2.3 sollen die vorgestellten Methoden anhand konkreter Beispiele des aktuellen Stand der Technik verglichen werden, um die Notwendigkeit des in der Einführung vorgestellten Ansatzes (1.2) aufzuzeigen.

2.1. Generierung von Benutzeroberflächen für Dienste

Die einfachste und zugleich aufwendigste Methode eine Benutzeroberfläche zur Dienstinteraktion zu erstellen besteht in der manuellen Programmierung einer Client-Softwarekomponente für einen bestimmten Dienst, beziehungsweise mehrere Dienste. Diese Methode hat neben der aufwendigen Entwicklung den Nachteil, dass zur Unterstützung weiterer Dienste oder bei Änderung eines unterstützten Dienstes eine manuelle Anpassung der Client-Softwarekomponente nötig ist. Daher wird im Folgenden nur die dynamische Generierung einer Benutzeroberfläche betrachtet.

Die dynamische Generierung von Benutzeroberflächen für Dienste bezeichnet ein Verfahren, das zur Laufzeit für beliebige Dienste anhand derer standardisierter Dienstbeschreibung automatisch eine graphische Benutzeroberfläche generiert. Dienstbeschreibungen werden mittels Dienst-Beschreibungssprachen wie zum Beispiel WSDL[1] im Umfeld der Service-orientierten Architekturen oder WADL[2] im Umfeld der Ressourcenorientierten Architekturen spezifiziert und dienen der externen Repräsentation eines Dienstes, indem Informationen zu Operationen, Aufrufparameter, Rückgabeparametern, Datentypen und weiteren Schnittstellenbeschreibungen eines Dienstes veröffentlicht werden. Anhand dieser öffentlichen Informationen, die jeder Dienst bereitstellt, lässt sich bereits durch Anwendung eines

[1] Web Services Description Language
[2] Web Application Description Language

einfachen Inferenzmechanismus eine Benutzeroberfläche generieren, wie in Abbildung 2.1 verdeutlicht wird.

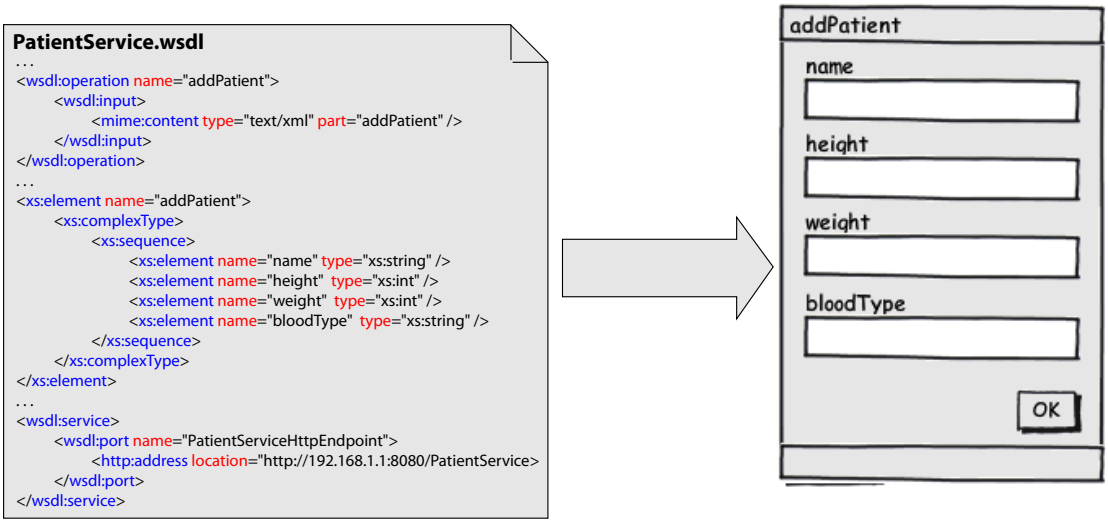


Abbildung 2.1.: Ableitung einer Benutzeroberfläche aus der Dienstbeschreibung

Das Beispiel zeigt Auszüge einer WSDL-Dienstbeschreibung für den Dienst „PatientService“, welche ausreichend sind, um eine graphische Benutzeroberfläche zum Aufruf der Dienst-Operation „addPatient“ zu erzeugen. Die Operation benötigt demnach als Eingabe die Parameter „name“, „height“, „weight“ und „bloodGroup“ für die Eingabefelder mit den jeweiligen Parameternamen als Bezeichner generiert werden können. Als Überschrift beziehungsweise Bezeichner der zu generierenden Benutzeroberfläche kann beispielsweise der Name der Operation verwendet werden. Um einen entfernten Dienstaufruf durchführen zu können, wird zusätzlich noch eine Schaltfläche zum Auslösen des Dienstaufrufes erzeugt. Die für den Dienstaufruf benötigte Adresse kann der Dienstbeschreibung unter den Definitionen des WSDL-Elements „service“ entnommen werden.

Wie am Beispiel zu erkennen, weist das Resultat der generierten Benutzeroberfläche jedoch starke Mängel auf. Da für die Bezeichner der Benutzerschnittstellenelemente nur die Bezeichner aus der funktionalen Schnittstellenbeschreibung zur Verfügung steht, ist die Verständlichkeit der generierten Benutzeroberflächen in vielen Fällen stark eingeschränkt. Zudem ist somit auch die natürliche Sprache der generierten Benutzeroberfläche auf diese Bezeichner festgelegt.

2.2. Generierung erweiterter Benutzeroberflächen für Dienste mittels Dienstannotationen

Um die Einschränkungen des Generierungsverfahrens einer Benutzeroberfläche anhand der Dienstbeschreibung aufzuheben, werden für das Generierungsverfahren zusätzliche semantische Informationen benötigt, die als Dienstannotationen bezeichnet werden. Dienstan-

notationen stellen Informationen zur Verfügung, die im Generierungsverfahren verwendet werden, um sowohl auf die Darstellung, als auch auf das Verhalten der generierten Elemente einer Benutzeroberfläche Einfluss nehmen zu können.

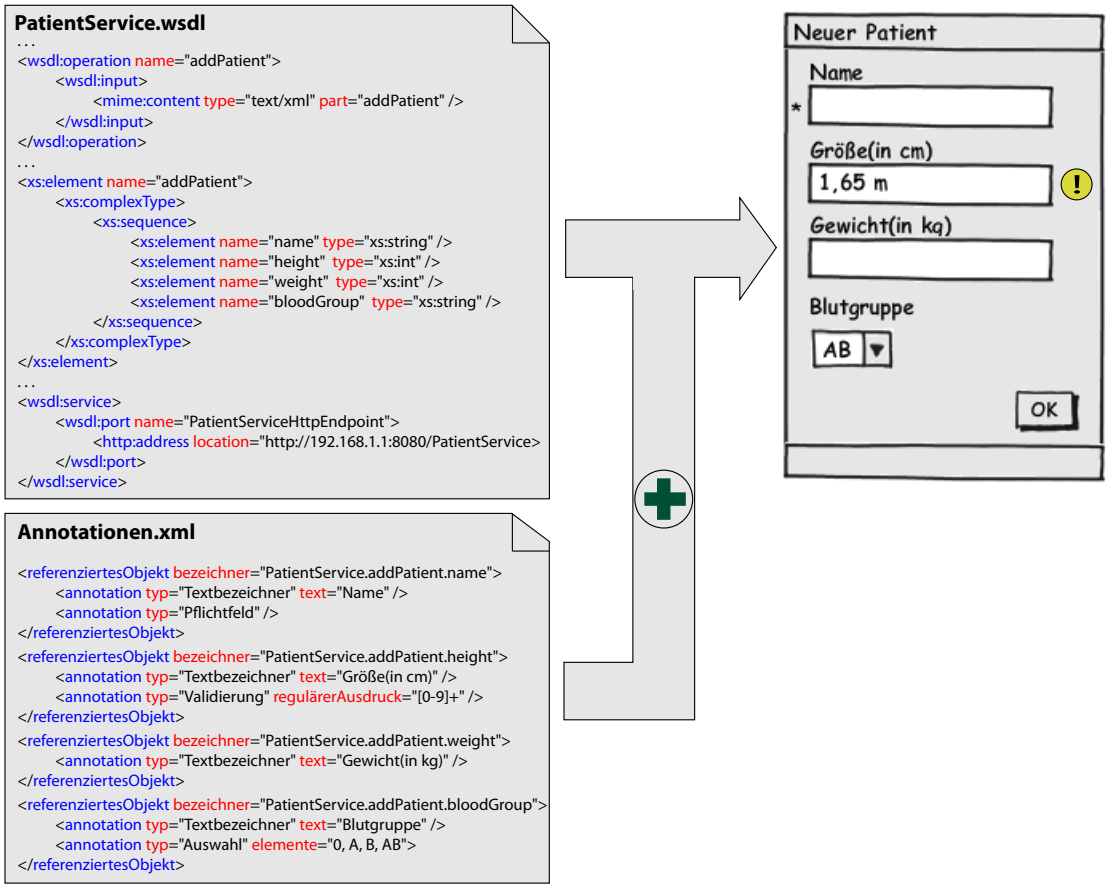


Abbildung 2.2.: Ableitung einer Benutzeroberfläche aus Dienstbeschreibung und Dienstannotationen

Das Beispiel in Abbildung 2.2 ergänzt das Beispiel aus Abschnitt 2.1 und zeigt neben der Dienstbeschreibung eine weitere XML-Datei mit Dienstannotationen, die in das Generierungsverfahren einbezogen werden und zum dargestellten Resultat führt. Da die Dienstannotationen im Beispiel nicht direkt in die WSDL-Datei der Dienstbeschreibung aufgenommen werden, sondern in einer separaten Datei abgespeichert sind, wird über das Element „referenziertesObjekt“ zunächst eine Möglichkeit der Referenzierung von annotierbaren Elementen der Dienstbeschreibung zur Verfügung gestellt. Dadurch ist es möglich Dienste, Operationen, Parameter und Datentypen einer Dienstbeschreibung zu adressieren und Dienstannotationen zuzuordnen. Neben der individuellen Anpassung der Bezeichner von Eingabeparametern, welche mittels der Dienstannotation „Textbezeichner“ realisiert wurde, stellt die Dienstannotation „Auswahl“ die Möglichkeit zur Verfügung, die Eingabe eines Parameters auf eine festgelegte Auswahl von Eingabewerten zu beschränken. Dadurch kann für die Eingabe des Parameters „bloodGroup“ ein Auswahlfeld generiert werden. Auf das Verhalten der Eingabefelder wird im Beispiel über die Dienstannotationen „Pflichtfeld“ und „Validierung“ Einfluss genommen. Während für ein Pflichtfeld ein entsprechender Hin-

weis erstellt werden kann, bietet die Dienstannotation „Validierung“ eine Möglichkeit die Eingabe anhand eines regulären Ausdrucks zu validieren.

Das Resultat zeigt, dass Dienstannotationen einen erheblichen Einfluss auf die Benutzbarkeit der generierten Benutzeroberfläche nehmen können, da dem Generierungsverfahren Informationen zur Verfügung gestellt werden, die zur Anwendung erweiterter Benutzeroberflächen-Funktionalitäten genutzt werden können. Dadurch besteht die Möglichkeit, die Generierung in einer beliebigen nativen Benutzerschnittstellensprache, durch Dienstannotationen auch mit deren spezifischen Funktionalitäten, zu unterstützen und auf diese Weise das für den Nutzer bekannte „Look and Feel“ bereitzustellen. Um eine kompfortable Erstellung von Dienstannotationen zu gewährleisten, sollte ein speziell für die Menge der unterstützten Dienstannotationen angepasster Editor zur graphischen Beschreibung bereitgestellt werden.

Bezüglich der Interaktionsmöglichkeiten mit einem Dienst bietet das vorgestellte Verfahren die Möglichkeit, nach der Wahl einer Operation des Dienstes einen Operationsaufruf über die für die Operation generierte Benutzeroberfläche durchzuführen. Mit der Rückgabe des Operationsaufrufs endet die Interaktion. Es ist demnach nicht möglich, erweiterte Interaktionsmöglichkeiten, die durch individuelle Navigations- und Datenflüsse beschrieben sind, zwischen Operationen eines Dienstes, beziehungsweise verschiedener Dienste, zu gestalten.

Zur Erzeugung interaktiver Anwendungen, die auf Diensten basieren, muss derzeit noch auf die manuelle Entwicklung von Client-Softwarekomponenten zurückgegriffen werden. Eine Generierung von interaktiven Anwendungen auf Basis von Diensten wurde bisher kaum erforscht und steht daher im Fokus dieser Arbeit.

Eine interaktive Anwendung zeichnet sich vor allem durch eine individuelle Anwendungsstruktur aus, die dem Nutzer das Navigieren zwischen verschiedenen Sichten einer Anwendung ermöglicht. Die einzelnen Sichten einer Anwendung stellen dann weitere Interaktionsmöglichkeiten zur Verfügung. In dieser Ausarbeitung wird für die Generierung von interaktiven Anwendungen der annotationsbasierte Ansatz aufgegriffen und erweitert. Neben Dienstannotationen, die zur erweiterten Beschreibung der Benutzeroberfläche dienen, werden zusätzliche Dienstannotationen eingeführt, die zur Definition von Navigations- und Datenflüssen sowie weiterer anwendungsrelevanter Eigenschaften genutzt werden können. Um die Anwendungserstellung nicht auf die Funktionalität eines Dienstes zu beschränken, sondern eine auf mehreren Diensten basierende Anwendung zu realisieren, wird der Annotationsmechanismus aus Beispiel 2.2 um die Möglichkeit der Referenzierung annotierbarer Dienstelemente beliebiger Dienste erweitert. Auf diese Weise kann jeder Sicht der zu generierenden Anwendung eine Dienstoperation bzw. eine Menge von Dienstoperationen zugeordnet werden. Navigationen zwischen Sichten können mit Hilfe der Navigationsdienstannotation definiert werden.

Der Ansatz beschreibt ein Verfahren, welches es ermöglicht, interaktive dienstbasierte Anwendungen durch Dienstannotationen plattformunabhängig zu beschreiben. Eine Generie-

rungskomponente kann diese Anwendungsbeschreibung auswerten, um Anwendungen für verschiedene Zielplattformen zu erzeugen.

2.3. Stand der Technik

Im Forschungsumfeld der Benutzerschnittstellengenerierung für Dienste wurden in den letzten Jahren unterschiedliche Konzepte entworfen und realisiert. Während im ersten Teil der Stand der Technik Analyse zunächst ausgewählte abstrakte Benutzerschnittstellenbeschreibungsformate dieser Konzepte und Umsetzungen vorgestellt werden, werden im zweiten Teil bestehende Editoren zum Erzeugen von Dienstannotationen betrachtet. Abschließend sollen durch eine Bewertung Vorteile, Nachteile und Grenzen der vorgestellten Beschreibungsformate und Editoren herausgestellt werden.

2.3.1. Annotationsmodelle und Beschreibungsformate

2.3.1.1. GUIDD

Das XML-Beschreibungsformat GUI Deployment Descriptor(GUIDD) wurde im Rahmen der Projekte von WSGUI [KKM03] und deren Erweiterung in Form des Dynvokers [Spi06a] entwickelt. Die Konzepte von WSGUI stellen mit der Veröffentlichung im Jahr 2003 einen ersten Ansatz vor, für Dienste auf Basis der funktionalen Dienstbeschreibung und einer optionalen Beschreibung in Form eines GUIDD, eine XForms[3]-basierte Benutzeroberfläche zu generieren. Die GUIDD-Beschreibungen dienen dazu, Dienstbeschreibungen zu ergänzen und zu erweitern, um dem Generierungsmechanismus genügend Informationen zur Generierung verständlicher Benutzeroberflächen zur Verfügung zu stellen.

Eine GUIDD-Datei ist immer einer Dienstbeschreibung zugeordnet und gliedert sich in die folgenden vier Abschnitte:

1. **Operations**

   Im Abschnitt Operations können Operationen der Dienstbeschreibung referenziert und mit zusätzlichen Informationen beschrieben werden. Dazu zählen die Angabe eines verständlichen Namens, die Beschreibung und das Festlegen der Bezeichnung der Schaltfläche zum Aufrufen der Operation.

2. **FormComponents**

   Über FormComponents können die einzelnen Elemente einer Dienstbeschreibung, welche Teil der Eingabemaske sind, detaillierter beschrieben werden. So können beispielsweise die Bezeichner von Eingabefeldern mit verständlichen Namen für verschiedene Sprachen versehen werden.

[3] http://www.w3.org/MarkUp/Forms/

3. **OutputTypes**

   Da die von WSGUI angewandte Formular-Beschreibungssprache XForms die Rückgabe von Operationsaufrufen nicht behandeln kann, bietet der Abschnitt OutputTypes die Möglichkeit Rückgabewerte mit MIME-Typen zu versehen um Nicht-Text-Rückgaben wie Bilder, Audio- oder Videodateien korrekt darstellen zu können.

4. **Stylesheets**

   Im Abschnitt Stylesheets können Cascading Style Sheets(CSS) eingebunden werden, um standardisierte Formatierungsanweisungen für die Generierung von webbasierten Benutzeroberflächen einzubeziehen.

Das GUIDD-Beschreibungsformat ermöglicht es, Dienstbeschreibungen auf einfache Weise zu ergänzen, um für einen Dienst eine verständliche Benutzeroberfläche in verschiedenen Sprachen zu erzeugen. GUIDD eignet sich besonders gut für die Anwendung einer dynamischen Benutzerschittstellengenerierung wie sie mit dem Dynvoker bereits erfolgreich umgesetzt wurden, da die GUIDD-Beschreibungen sehr einfach in den Inferenzmechanismus einbezogen werden können. Da GUIDD in der aktuellen Spezifikation die Anwendung von nur wenigen Dienstannotationen ermöglicht, ist der Beschreibungsumfang allerdings stark eingeschränkt.

2.3.1.2. ServFace Annotation Model

Das ServFace Annotation Model wurde im Rahmen des ServFace-Projektes[Pro09] entwickelt. Das ServFace-Projekt verfolgt das Ziel interaktive dienstbasierte Anwendungen auf einfache Weise über werkzeuggestützte Frontend-Komposition zu beschreiben und für verschiedene Zielplattformen generieren zu können. Das ServFace Annotation Model wird dabei für den ersten Schritt benötigt, um eine umfangreiche Sammlung von Dienstannotationen für den Generierungsprozess von Benutzeroberflächen für Dienste bereitzustellen. Im Modell wird zwischen drei unterschiedlichen Gruppen von Dienstannotationen, welche im Folgenden kurz vorgestellt werden, unterschieden. Da einige Dienstannotationen jedoch Eigenschaften verschiedener Gruppen vereinen und eine strikte Gruppierung der Dienstannotationen daher kaum möglich ist, handelt es sich vielmehr um eine Gruppierung:

1. **Visuelle Annotationen**

   Visuelle Annotationen haben immer einen direkten Einfluss auf die visuelle Erscheinung der generierten Benutzeroberfläche, indem Einfluss auf die Art beziehungsweise Eigenschaftswerte der zu generierenden Benutzerschnittstellenelemente genommen wird. Die Gruppierungsanweisung von Elementen, das Festlegen von Bezeichnernamen in verschiedenen Sprachen und das Definieren einer festgelegten Auswahl für ein Eingabefeld stellen die bekanntesten Vertreter dieser Gruppe dar.

2. **Verhaltensorientierte Annotationen**

   Zu den verhaltensorientierten Annotationen zählen Beschreibungen, die für Ein- beziehungsweise Ausgabefelder bestimmte Definitionen festlegen, um Benutzerschnittstellenfunktionalitäten aktueller graphischer Benutzerschnittstellenbibliotheken zu ermöglichen. Hierzu zählen beispielsweise Validierung, Eingabeempfehlungen, automatische Formularvervollständigung oder die Angabe eines Defaultwerts für ein Eingabefeld.

3. **Relationale Annotationen**

   Relationale Annotationen beschreiben Beziehungen zwischen Elementen der Dienstbeschreibung. So können beispielsweise Beziehungen zwischen einem Parameter und einem komplexen Datentyp abgebildet, oder der Aufruf einer Dienstoperation von einer erfolgreichen Authentifizierung über eine Authentifizierungsoperation abhängig gemacht werden.

Zusätzlich zu den definierten Dienstannotationen gibt es mit Hilfe der Annotation „Custom-Annotation“ die Möglichkeit den Beschreibungsumfang des ServFace Annotation Models zu erweitern. Um einen Referenzmechanismus zum Abbilden von Dienstannotationen auf annotierbare Elemente einer Dienstbeschreibung anwenden zu können, wurde ein abstraktes Dienstmodell beschrieben.

Das ServFace Annotation Model bietet mit seinen vielfältigen Dienstannotationen sehr umfangreiche Beschreibungsmöglichkeiten. Obwohl im Modell relationale Dienstannotationen unterstützt werden und das Ziel von ServFace darin besteht interaktive Anwendungen zu erzeugen, kann mit dem Annotations-Modell alleine noch keine Anwendung beschrieben werden, da beispielsweise Navigations- und Datenflüsse nicht ausgedrückt werden können. Im Ablauf der ServFace-Methodik werden diese anwendungsrelevanten Informationen erst in einem weiteren Modell auf einer plattformabhängigen Ebene beschrieben.

2.3.1.3. Kawash

Im Jahr 2004 hat Kawash ein Verfahren zur dynamischen Generierung von Benutzerschnittstellen für Dienste vorgestellt [Kaw04]. Mit diesem können mit Hilfe einer annotationsbasierten Beschreibungssprache benutzerdefinierte, browserbasierte Webanwendungen für unterschiedliche Zielplattformen beschrieben werden. Das Ziel von Kawash war es, angepasste Benutzerschnittstellen zum Aufruf von Diensten für mobile Endgeräte mit unterschiedlichen Anzeigeeigenschaften zu beschreiben. Die Beschreibung von Benutzerschnittstellen wird nach diesem Verfahren in zwei unterschiedliche Bereiche unterteilt: *Benutzerschnittstellenfunktionalität* und *Benutzerschnittstellenelementattribute*, wobei beide Beschreibungen durch eine XML-Spezifikation formuliert und genau einem Dienst zugeordnet werden.

Benutzerschnittstellenfunktionalität

Die Funktionalität der zu generierenden Benutzerschnittstellen wird mittels eines erweiterten endlichen Automaten beschrieben. Dieser ermöglicht die vollständige Abbildung einer Anwendungsstruktur samt Benutzeroberflächenbeschreibungen und wird durch das Dreitupel $(\mathcal{S}, \mathcal{T}, \mathcal{M})$ spezifiziert.

- **$\mathcal{S}$ - Anwendungszustände**

  $\mathcal{S}$ beschreibt eine Menge von Anwendungszuständen, wobei jeder Zustand durch eine Menge von Benutzerschnittstellenelementen, welches Ein- und Ausgabefelder für Dienstparameter oder Kontrollelemente zum Auslösen von Ereignissen sein können, beschrieben ist und einer Darstellungsseite der Webanwendung entspricht. Es gibt einen Start- beziehungsweise einen Endzustand, welcher den Ein- beziehungsweise Austrittspunkt der Anwendung kennzeichnet.

- **$\mathcal{T}$ - Transitionsfunktion**

  Die Transitionsfunktion $\mathcal{T}$ auf $\mathcal{S}$ definiert alle mögliche Übergänge zwischen Anwendungszuständen. Transitionen können sowohl an Aufrufe durch den Nutzer, als auch an Systemereignisse gebunden werden.

- **$\mathcal{M}$ - Dienstabbildungen**

  Dienstabbildungen stellen den Bezug von Benutzerschnittstellenelementen zu den Elementen der Dienstbeschreibung her, um für einen Dienstaufruf entsprechende Felder der Benutzeroberfläche mit den Ein- und Ausgabeparametern zu verknüpfen. Weiterhin besteht die Möglichkeit bedingte Parameterabbildungen von dem Rückgabewert eines Dienstaufrufes zu Eingabefeldern des nachfolgenden Anwendungszustands zu beschreiben. Es werden allerdings nur Bindungen auf Parameterebene unterstützt. Eine Abbildung auf Schemainstanzen der funktionalen Dienstbeschreibung ist nicht möglich.

Attribute von Elementen der Benutzeroberfläche

Obwohl durch die Beschreibung der *Benutzerschnittstellenfunktionalität* bereits alle relevanten Aspekte für eine dynamische Generierung der Benutzeroberflächen abgedeckt sind, bietet die zusätzliche Beschreibung von Elementattributen erst die Möglichkeit auf Eigenschaften der zu generierenden Benutzerschnittstellenelemente Einfluss zu nehmen. So können beispielsweise Gruppierungen von Benutzerschnittstellenlementen und benutzerdefinierte Bezeichner für Eingabefelder und Schaltflächen festgelegt werden. Aber auch erweiterte Funktionalitäten wie das Ersetzen eines Eingabefeldes mit einer Auswahlbox oder das Festlegen eines Pflichtfeldes, wie sie bereits vom ServFace-Annotationmodel bekannt sind, können beschrieben werden. Die Anwendung der beschriebenen Benutzerschnittstellenfunktionalitäten hängt von den Fähigkeiten des jeweils eingesetzten mobilen Endgerätes ab.

Das von Kawash vorgestellte Verfahren eignet sich gut, um maßgeschneiderte Benutzerschnittstellen für Dienste zu erzeugen. Das eingeführte Beschreibungsformat ermöglicht

es Navigations- und Datenflüsse zu formulieren, unterscheidet sich aber aufgrund der Beschreibungseigenschaften von herkömmlichen Dienstannotationen. Da für die Benutzerschnittstellenbeschreibung nur auf die Funktionalität eines zugewiesenen Dienstes zurückgegriffen werden kann und nur browserbasierte Benutzerschnittstellen erzeugt werden, ist das Verfahren bezüglich der Entwicklung interaktiver, dienstbasierter Anwendungen eingeschränkt.

2.3.1.4. Khushraj, Lassila

Die Einführung von semantischen Webservices ermöglicht es, Dienste durch zusätzliche Informationen, die nicht durch funktionale Dienstbeschreibungen ausgedrückt werden können, zu beschreiben, um dem Nutzer eine detailliertere Beschreibung zum Inhalt und zu den Funktionalitäten von Diensten zu bieten. Mit OWL-S [Bur04] wurde 2004 eine W3C-Spezifikation zur semantischen Auszeichnung von Diensten veröffentlicht. Die Beschreibungssprache ermöglicht semantische Informationen zum Auffinden, Ausführen, Zusammensetzen und Überwachen von Diensten zu formulieren.

Khushraj und Lassila griffen diese erweiterte Beschreibungssprache für Dienste auf und stellten 2005 im Rahmen einer Forschungsarbeit für das Nokia Forschungszentrum in Burlington (USA) ein Verfahren vor, um die Qualität automatisch generierter Benutzeroberflächen für Dienstaufrufe zu verbessern. [KL05] Hierzu wurde die Ontologie von OWL-S zum einen um die Möglichkeit der Beschreibung von Annotationen für Benutzerschnittstellenelemente erweitert, wodurch folgende Ausdrucksmöglichkeiten gegeben sind:

- benutzerdefinierte Bezeichner für Eingabefelder
- Festlegen bevorzugter Typen für Eingabefelder(z.B. Texteingabe, Checkbox)
- Definition von geordneten Auswahllisten für Eingabefelder
- Gruppierung von Benutzerschnittstellenelementen

Die Informationen dieser erweiterten Beschreibung werden durch ein *UIModel* ausgedrückt. Das UIModel bietet die Möglichkeit, Annotationen auf Elemente der funktionalen Dienstbeschreibung abzubilden und ist genau einem Dienst zugeordnet. Bei einem Aufruf eines Dienstes mit erweiterter OWL-S Beschreibung wird das UIModel von einer Generatorkomponente geladen und in den Prozess der Benutzerschnittstellengenerierung eingebunden.

Um die Qualität der zu generierenden Benutzerschnittstellen weiter zu verbessern, wurde von Khushraj und Lassila das *Semantic Caching* eingeführt. Es sieht vor, lokal verfügbare Informationen über den Nutzer zu sammeln und zur weiteren Verbesserung der Benutzerschnittstelle einzubinden. Ziel ist es, semantische Beziehungen zwischen einem Ein- oder Ausgabefeld und der durch das Feld beschriebenen Daten zu erkennen und folglich eine automatische Anpassung zu ermöglichen. Für das Semantic Caching werden beispielsweise folgende Nutzerdaten verwendet:

- Nutzerprofilinformationen

- PIM-Informationen wie Kalender- oder Adressbucheinträge
- Kontextinformationen
- Verlauf vorheriger Eingabearten bei Dienstaufrufen
- Firmendaten und Telefonbucheinträge

Mit Hilfe dieser zusätzlichen Informationen über den Nutzer können beispielsweise geeignete Standardwerte oder Auswahllisten für Eingabefelder festgelegt oder Eingabefelder komplett entfernt werden, wenn sich der Wert der Eingabe bereits aus den Nutzerinformationen ableiten lässt.

Das vorgestellte Beschreibungsformat ermöglicht durch die Erweiterung des OWL-S Profils zur Benutzerschnittstellenbeschreibung durch nur wenige Annotationen bereits eine deutliche Verbesserung der Benutzerfreundlichkeit von generierten Eingabeoberflächen.

2.3.2. Editoren

2.3.2.1. WSGUI-Editor

Der WSGUI-Editor wurde im Rahmen einer Diplomarbeit entwickelt [Spi06b], um eine werkzeuggestützte Erstellung von WSGUI-Hinweisen für einen Dienst zu ermöglichen und somit das Resultat einer dynamischen Generierung der Benutzeroberflächen zu verbessern. Die Bearbeitung mit dem Editor beginnt mit dem Import einer WSDL-Dienstbeschreibung. Nach dem Import stellt der Editor Möglichkeiten zum Hinzufügen von WSGUI-Hinweisen in unterschiedlichen Ansichten zur Verfügung und über eine Vorschaukomponente kann das Ergebnis während des Editierprozesses jederzeit kontrolliert werden. Die während einer Bearbeitung erstellten WSGUI-Hinweise werden als GUIDD-Datei abgespeichert und können beispielsweise vom Dynvoker zur dynamische Erzeugung verständlicher Benutzeroberflächen für Dienste in den Generierungsprozess eingebunden werden.

Der WSGUI-Editor vereinfacht das Annotieren von Diensten durch komfortable Editiermöglichkeiten. Da als Beschreibungsformat GUIDD verwendet wird, ist der Annotationsumfang allerdings auf die im Abschnitt 2.3.1.1 beschriebenen Möglichkeiten beschränkt.

2.3.2.2. Annotation-Tool

Das Annotation-Tool ([IJH+09]) soll den ersten Schritt der ServFace-Methodik [Jug10] unterstützen, indem zu je einer Dienstbeschreibung über eine graphische Benutzeroberfläche ServFace-Dienstannotationen erstellt werden können. Zum Editieren stellt das Werkzeug eine Dienstansicht bereit, die das Selektieren einzelner Dienstelemente erlaubt. Über mehrere Eingabemasken können dann im Editorbereich Dienstannotationen für das selektierte Dienstelement erstellt werden. Die erstellten Dienstannotationen werden zur Veröffentlichung für die weiteren ServFace-Werkzeuge ServFace Builder (2.3.2.3) und Advanced Composition Tool (2.3.2.4) als XML-Instanz des ServFace Anotation Model (2.3.1.2) in

ein zentrales Repository abgespeichert und können von dort auch wieder importiert und weiterbearbeitet werden.

Das Annotation-Tool erleichtert die Arbeit für einen Annotator im Vergleich zur manuellen Erstellung von Dienstannotationen enorm, da er mit dem Werkzeug über wenige Eingabemasken alle ServFace-Dienstannotationen beschreiben kann. Auch der Annotationsumfang ist aufgrund des eingesetzten ServFace Annotation Models deutlich mächtiger als der des WSGUI-Editors und ermöglicht so eine detaillierte Benutzerschnittstellenbeschreibung. Die erstellten ServFace-Dienstannotationen stellen in der ServFace-Methodik nur einen Zwischenschritt hin zur Erstellung interaktiver, dienstbasierter Anwendungen dar. Daher werden die Dienstannotation für die Generierung von Benutzeroberflächen interaktiver, dienstbasierter Anwendungen berücksichtigt, es gibt derzeit aber keine veröffentlichte Inferenz-Engine die eine dynamische Benutzerschnittstellengenerierung für Dienste nur unter Einbeziehung von ServFace-Dienstannotationen unterstützt. In einer Belegarbeit [Mar09][4] wurde aber für die Google-Android[5] Plattform prototypisch ein Interpreter umgesetzt, der eine ad-hoc Integration von Diensten, die mit einer Teilmenge der ServFace-Dienstannotationen annotiert sein können, auf mobilen Endgeräten ermöglicht.

2.3.2.3. ServFace-Builder

Der Servface-Builder ist ein im Rahmen des ServFace-Projekts entwickelter web-basierter Editor zum Erzeugen interaktiver, dienstbasierter Anwendungen. Der Editor setzt den Ansatz einer Dienstkomposition auf Präsentationsebene um, indem homogene Anwendungen alleine durch die Beschreibung einer gemeinsamen Benutzeroberfläche aus verschiedenen Diensten modelliert werden können. Eine Zielstellung des Projektes war es, die Nutzung des Editors möglichst einfach zu gestalten, so dass auch Nutzer ohne Vorkenntnisse im Umfeld von Dienstkompositionen zielgerichtet Anwendungen erstellen können. Zu diesem Zweck werden die mit dem Annotation-Tool vorab erstellten Dienstannotationen vom ServFace-Builder genutzt, um während des Editierprozesses eine Vorschau der Benutzeroberfläche darzustellen. Abbildung 2.3 gibt einen Überblick über den Aufbau des Editors.

Der Nutzer hat die Möglichkeit, beliebig viele Seiten zur Beschreibung einer Anwendung anzulegen. Diese können in der Übersichtsansicht frei angeordnet und mit Transitionen versehen werden. So wird ausgehend von einem Startzustand die gesamte Navigationsstruktur der editierten Anwendung dargestellt. Um einzelne Seiten detaillierter zu beschreiben, können diese geöffnet werden, was zu einer vergrößerten Darstellung der Seite als Vorschau der Nutzerschnittstellenelemente führt. In diesem Darstellungsmodus besteht die Möglichkeit, die Seite durch Funktionsblöcke, welche jeweils zur Darstellung der Benutzeroberfläche eines Operationsaufrufes dienen, zu erweitern und Beziehungen zwischen einzelnen Eingabefeldern einer Seite zu beschreiben, um nutzerfreundlichere Eingabefunktionen zu ermöglichen.

[4] Erkenntnisse der Arbeit flossen in die Konzeption des Gesamtansatzes und in die Technologieevaluierung ein.
[5] http://www.android.com/

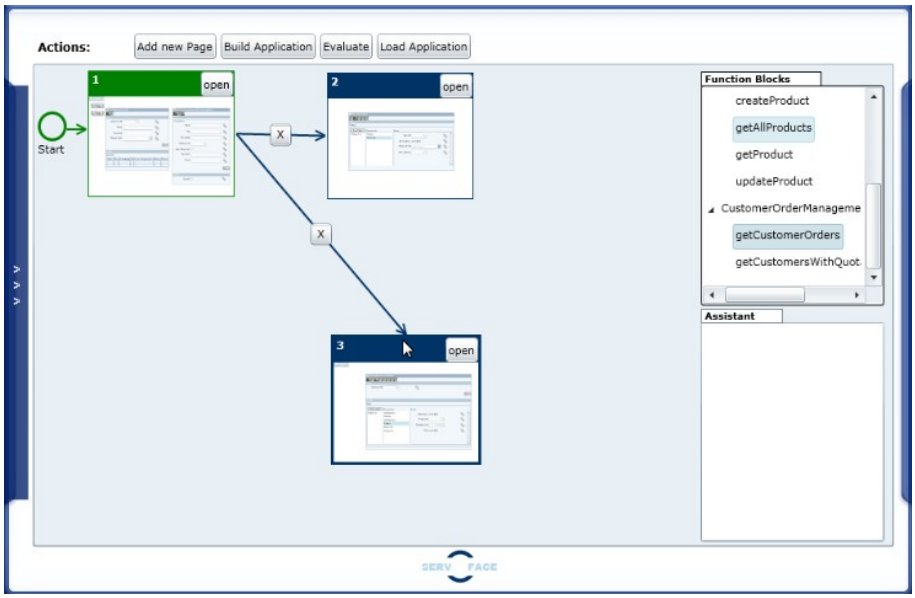


Abbildung 2.3.: ServFace-Builder

Die modellierte Anwendung wird durch das Composite Application Model (CAM) beschrieben und als Instanz dieses Modells vom Editor exportiert. Hierbei handelt es sich um ein Anwendungsmodell, welches den gesamten Aufbau der Anwendung durch Seiten, Navigationsflüsse, Datenflüsse und Interaktionselementen, welche auf Benutzerschnittstellenelemente abbilden, beschreibt.

2.3.2.4. Advanced Composition Tool

Mit dem Advanced Composition Tool wurde ein zweiter Editor zur Beschreibung von interaktiven, dienstbasierten Anwendungen im Rahmen des ServFace-Projektes erstellt. In Kontrast zum ServFace-Builder richtet sich dieses Softwarewerkzeug an erfahrene Anwendungsdesigner und ermöglicht eine deutlich feingranularere Anwendungsbeschreibung. Um dies umzusetzen wird die Benutzerschnittstellenbeschreibungssprache Maria XML [PSSD09] eingesetzt. Maria XML ermöglicht es eine Anwendung auf unterschiedlichen Abstraktionsebenen zu beschreiben und ist an das Cameleon Referenzmodell [**?**] angelehnt. Diese Eigenschaft wurde für die Entwicklungsmethodik mit dem Editor ausgenutzt, indem eine iterative Anwendungsbeschreibung in mehreren Phasen nach folgendem Vorgehen unterstützt wird:

1. Die Erzeugung eines Aufgabenmodells mittels der *Concurr Task Tree* Notation [CTT]; Diese ermöglicht die gesamte Anwendung durch eine Menge von Anwendungsaktivitäten zu beschreiben. Zwischen den Aktivitäten können mit dem Aufgabenmodell auch temporalen Beziehungen und Abhängigkeiten definiert werden.

2. Bindungen zwischen Aktivitäten und Dienstoperationen herstellen, um Dienstaufrufe mit festgelegten Systemereignissen zu verbinden;

3. Erzeugung eines ersten abstrakten und plattformunabhängigen Modells in Maria XML aus den Informationen des Aufgabenmodells; Hierbei werden bestehende ServFace-Dienstannotationen von verwendeten Diensten berücksichtigt und können manuell angepasst werden. Das durch diesen Prozess erzeugte Modell beinhaltet bereits notwendige Informationen, um Eingabemasken darzustellen, Dienstaufrufe durchzuführen und Rückgabenachrichten anzuzeigen.

4. Verfeinerung der Benutzerschnittstellenbeschreibung durch ein konkretes und plattformabhängiges Modell in Maria XML; Das Modell kann detaillierte Benutzerschnittstellenfunktionalitäten für verschiedene Plattformen abbilden.

5. Weitere Verfeinerungen des konkreten Modells auf Implementierungsebene für spezielle Benutzerschnittstellensprachen und Plattformen;

Nachdem die Anwendungsbeschreibung abgeschlossen ist, erfolgt die Generierung einer ausführbaren Anwendung durch Modell-zu-Modell beziehungsweise Modell-zu-Code Transformation.

2.3.3. Zusammenfassung

Die vorgestellten Annotationsformate belegen, dass in den letzten Jahren mehrere Ansätze zur Benutzerschnittstellengenerierung für Dienste unter Einbeziehung von zusätzliche UI-Beschreibungen entstanden sind. Tabelle 2.1 gibt einen Überblick über die wichtigsten Eigenschaften der vorgestellten Annotationsformate.

Annotationsformat	**Navigationsbeschreibung**	**Beschreibungsumfang**
GUIDD	nein	gering
ServFace	nein	hoch
Kawash	ja	gering
Khushraj, Lassila	nein	mittel

Tabelle 2.1.: Eigenschaften der Annotationsformate

Bei einer zusammenfassenden Betrachtung werden folgende Einschränkungen deutlich:

- Der Beschreibungsumfang von den vorgestellten Annotationsformaten ist mit Ausnahme des ServFace Annotationmodels zu gering, um reichhaltige Benutzerschnittstellen zu beschreiben. Insbesondere die Beschreibung aktueller Benutzerschnittstellenfunktionalitäten, wie beispielsweise Eingabeempfehlungen oder eine Eingabevalidierung, werden nicht unterstützt.

- Die Bindung von Dienstannotationen an die entsprechenden Knoten der funktionalen Schnittstellenbeschreibung wurde unzureichend umgesetzt. Keines der vorgestellten Beschreibungsmodelle ermöglicht das Verweisen auf Schemainformationen einer WSDL-Beschreibung. Auch bieten die Beschreibungsformate, bis auf das ServFace

Annotationmodel, keine Möglichkeit zur Komposition mehrerer Dienste auf Frontend-Basis, da diese nur Knoten innerhalb ihrer zugehörigen Schnittstellenbeschreibung referenzieren können.

- Da die meisten Annotationsformate unter dem Aspekt der Benutzerschnittstellengenerierung zur Darstellung im Browser entwickelt wurden, können die Annotationen nicht zur Beschreibung von plattformspezifischen Informationen genutzt werden. Auch hier grenzen sich die ServFace Dienstannotationen von den anderen Formaten ab, da diese Kontextbindungen an Plattformen und Sprachen ermöglichen.

- Das Beschreibungsformat von Kawash bietet als einziges die Möglichkeit einen individuellen Navigationsfluss für die zu generierenden Benutzerschnittstellen zu formulieren. Dieser ist jedoch auf die Funktionalitäten nur einer zugeordneten funktionalen Schnittstellenbeschreibung beschränkt. Für die weiteren Formate ist der resultierende Navigationsfluss durch die Struktur der funktionalen Schnittstellenbeschreibung vordefiniert. Die Navigation erfolgt demnach immer von einer Eingabesicht zu einer Ausgabesicht einzelner Dienstoperationen.

Auch für das Erstellen von Dienstannotationen beziehungsweise dem Beschreiben von dienstbasierten Anwendungen, wurden im Rahmen verschiedener Ansätze Softwarekomponenten zum werkzeuggestützten Vorgehen umgesetzt. Tabelle 2.2 gibt einen Überblick über die wichtigsten Eigenschaften der vorgestellten Editoren.

Editor	**indiv. Anwendungsmodellierung**	**Exportformat**
WSGUI-Editor	nein	GUIDD
Annotation-Tool	nein	ServFace Annotationen
ServFace-Builder	ja	CAM
Advanced Composition Tool	ja	MARIA XML

Tabelle 2.2.: Eigenschaften der Annotationseditoren

Da von den vorgestellten Editoren nur der ServFace-Builder und das Advanced Composition Tool die individuelle Modellierung von interaktiven, dienstbasierten Anwendungen ermöglichen, und somit direkt mit dem zu realisierenden Softwarewerkzeug verglichen werden können, werden die anderen Editoren in der folgenden Beurteilung nicht betrachtet.

Der bedeutende Unterschied zu dem in dieser Arbeit verfolgten Ansatz besteht darin, dass die im Rahmen des ServFace-Projektes entstandenen Editoren die modellierten Anwendungen als separates Anwendungsmodell und nicht in Form von Dienstannotationen beschreiben. Dies führt zu folgenden Beeinträchtigungen:

- Die Modellierung einer Anwendung ist immer auf genau eine Plattform beschränkt, da das Anwendungsmodell plattformspezifische Informationen enthält. Um die selbe Anwendung oder eine leicht veränderte Anwendung auf einer weiteren Plattform zu veröffentlichen, muss eine weitere Instanz des Anwendungsmodells erstellt werden.

- Da für die Anwendungserstellung auch die Informationen der ServFace-Dienstannotationen berücksichtigt und ins Anwendungsmodell übertragen werden, ergeben sich komplexe Abhängigkeiten die berücksichtigt werden müssen, wenn die Dienstannotationen geändert werden. Außerdem entsteht durch das Übertragen der implizit in den Dienstannotationen vorliegenden Informationen in das Anwendungsmodell ein zusätzlicher Modellierungsoverhead.

Die aufgeführten Nachteile können vermieden werden, indem die gesamte Anwendungsbeschreibung einer interaktiven, dienstbasierten Anwendung mit Hilfe von Dienstannotationen ausgedrückt wird. Zu diesem Zweck bietet das ServFace-Annotationsmodell das größte Potential, da es einen hohen Beschreibungsumfang aufweist, das Beschreiben von Abhängigkeiten zur Zielplattform ermöglicht und über einen Dienstreferenzierungsmechanismus verfügt, der eine Frontend-Komposition von Diensten ermöglicht. Das Annotationsmodell muss allerdings um Dienstannotationen zur Beschreibung von Navigations- und Datenflüssen, sowie weiterer anwendungsrelevanter Funktionalitäten erweitert werden.

3. Analyse

In diesem Kapitel wird zunächst ein für die Umsetzung des Ansatzes geeignetes Anwendungsszenario vorgestellt. Anhand des entworfenen Anwendungsszenarios werden daraufhin identifizierte Anforderungen an das zu realisierende System beschrieben.

3.1. Anwendungsszenario

Das Szenario, welches im Folgenden beschrieben wird, wurde im Rahmen der Dissertation [Fel11] aufgestellt und für diese Ausarbeitung in einer vereinfachten Form übernommen.

Um die Arbeit eines Landarztes in stark entlegenen Regionen zu unterstützen, soll auf Basis von Diensten eine mobile Anwendung erstellt werden, die das Erfassen medizinischer Patientendaten und das Bestellen von Medikamenten beziehungsweise medizinischen Verbrauchsgütern ermöglichen soll. Die Anwendung soll auf einem mobilen Endgerät mit Internetzugang ausgeführt werden und dem Landarzt vor allem zur Datenerfassung bei Hausbesuchen dienen. Um die Anwendungsfunktionalität bereitzustellen, werden zwei Dienste benötigt. Ein Patientendienst bietet die notwendigen Operationen um nach Patienten zu suchen, einen neuen Patienten in das System aufzunehmen und Patientendaten zu editieren. Um Bestellungen von Medikamenten und medizinischen Verbrauchsgütern aufzunehmen, stellt ein Medikamentendienst erforderliche Operationen zur Verfügung. Abbildung 3.1 stellt die Struktur der zu realisierenden Anwendung mit allen Anwendungszuständen und Transitionen zwischen den Anwendungszuständen dar.

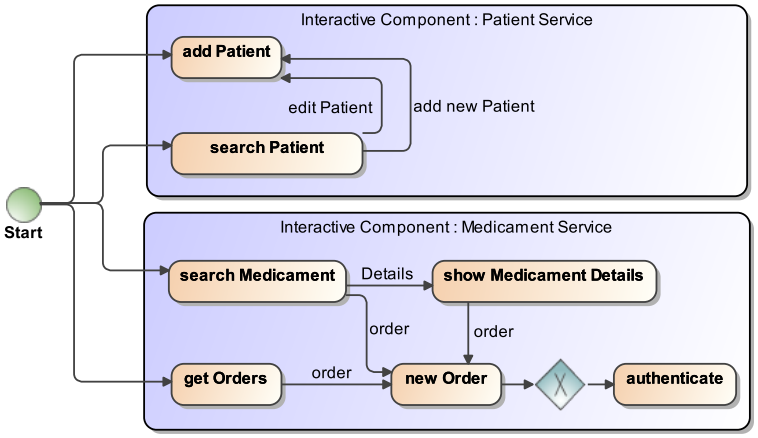


Abbildung 3.1.: Überblick über die Anwendungsstruktur

Demnach setzt sich die Anwendung aus zwei unabhängigen interaktiven Komponenten zusammen, die jeweils einem Dienst zugeordnet sind. Die interaktiven Komponenten beinhalten eine Menge von Anwendungszuständen, die aus den Dienstoperationen der zugeordneten Dienste abgeleitet wurden.

Nach dem Start der Anwendung wird dem Nutzer auf einer Startseite zunächst die Liste von initialen Operationen, die vom Startzustand der Anwendung erreichbar sind, zur Verfügung gestellt (3.2 Abbildung (a)). Der Anwendungsfall, welcher im Folgenden näher betrachtet werden soll, bezieht sich auf die Patientenverwaltung.

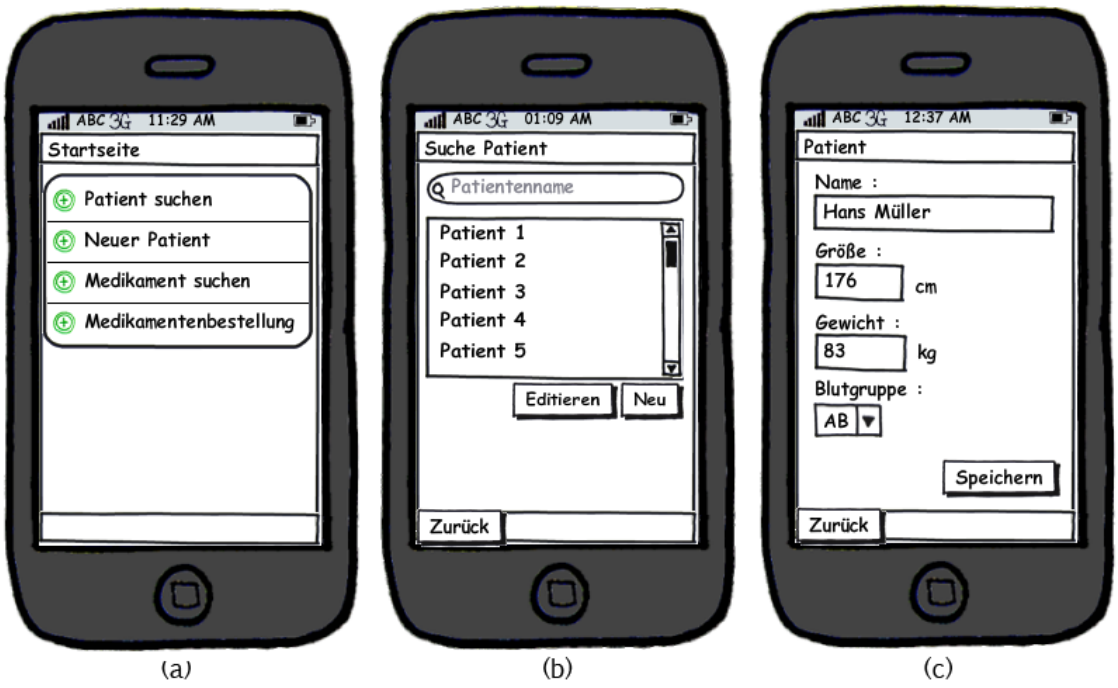


Abbildung 3.2.: Anwendungsszenario zur Patientenverwaltung

Von der Startseite als Ausgangspunkt der Anwendung hat der Nutzer zur Verwaltung von Patienten die Möglichkeit nach einem Patienten zu suchen oder einen neuen Patienten zu erfassen. Wählt der Nutzer die Möglichkeit nach einem Patienten zu suchen, wird die Benutzeroberfläche „Suche Patient“ aufgerufen(3.2 Abbildung (b)). Über das Sucheingabefeld kann hier durch Eingabe des Namens nach Patienten gesucht werden. Die Suchergebnisliste wird daraufhin noch auf der selben Benutzeroberfläche angezeigt und der Nutzer kann nach Selektion eines Patienten in der Suchergebnisliste über das zur Verfügung stehende Navigationselement „Editieren“ die Daten des Patienten bearbeiten. Sollte der gesuchte Patient nicht in der Suchergebnisliste aufgeführt sein besteht die Möglichkeit über das Navigationselement „Neu“ einen neuen Patienten in das System aufzunehmen. In beiden Fällen wird der Nutzer auf die Benutzeroberfläche „Patient“ weitergeleitet, wobei im Fall des Editierens die bestehenden Daten des Patienten in die dafür vorgesehenen Eingabefelder eingetragen sind(3.2 Abbildung (c)). Die Patientendaten können editiert beziehungsweise für einen neuen Patienten angelegt werden. Über das Navigationselement „Speichern“ wird der Patientendienst aufgerufen, um die Daten im System abzuspeichern. Wenn ein neuer Patient angelegt wird, werden neben den über die Eingabemaske eingegebenen Daten zu-

sätzliche Informationen über Standort (GPS-Koordinate) und Zeitpunkt übermittelt und mit den Patientendaten im System abgespeichert. Diese Informationen werden über Systemfunktionen des mobilen Endgerätes abgerufen und können später beispielsweise für eine Navigation des Landarztes zu einem Patienten genutzt werden.

3.2. Anforderungsanalyse

Aus dem vorgestellten Szenario lassen sich Anforderungen an das zu realisierende Softwarewerkzeug und an die zu konzipierenden Dienstannotationen ableiten, welche im Folgenden separat beschrieben werden.

Anforderungen an das Softwarewerkzeug

/AS1: Benutzbarkeit

Für die Anwendungserstellung soll eine Möglichkeit für das graphische Editieren der Dienstannotationen zur Beschreibung von Navigations- und Datenflüssen bereitgestellt werden. Das Softwarewerkzeug und insbesondere der graphische Editor soll eine fehlerfrei nutzbare und leicht zu erlernende Bedienung gewährleisten, so dass ein fachkundiger Nutzer nach einer kurzen Einführung in der Lage sein soll mit dem Softwarewerkzeug zielgerichtet eine Anwendung zu erstellen.

/AS2: Erweiterbarkeit und Modularität

Das Softwarewerkzeug soll so konzipiert werden, dass Änderungen und Erweiterungen des Systems einfach umgesetzt werden können. Durch einen modularen Aufbau, der zusätzlich die Wiederverwendbarkeit einzelner Komponenten und die Integration bestehender Komponenten unterstützt, kann die Erweiterbarkeit des Systems garantiert werden.

/AS3: Serialisierung einer erstellten Anwendung in ein festgelegtes Zielmodell

Um aus einer Anwendung, die mit dem Softwarewerkzeug erstellt wurde, eine lauffähige Anwendungen für eine beliebiger Zielplattformen zu erhalten, muss die erstellte Anwendung als Instanz eines festgelegten Annotationenmodells einem externen System für die weitere Generierung zur Verfügung gestellt werden. Zu diesem Zweck sollte das Softwarewerkzeug eine geeignete Exportfunktion in das Annotationenmodell zur Verfügung stellen. Darüber hinaus soll das Softwarewerkzeug die Möglichkeit bieten eine erstellte Anwendung in geeigneter Form abzuspeichern und diese zu einer späteren Weiterbearbeitung zu laden.

Anforderungen an die Dienstannotationen

/AD1: Beschreibung von Navigations- und Datenflüssen

Die Erstellung einer Anwendungsbeschreibung mit Navigations- und Datenflüssen stellt ein zentrales Ziel dieser Arbeit dar. Folglich sind zur Beschreibung der Navigations- und

Datenflüsse geeignete Dienstannotationen zu erstellen. Mit Hilfe dieser Annotationen sollen Übergänge zwischen den Seiten einer Anwendung beschrieben werden können, wobei Übergänge zusätzlich das Übermitteln von Daten in die Zielseite erlauben sollen.

/AD2: Beschreibung erweiterter Funktionalitäten, die mit ServFace-Annotationen nicht ausgedrückt werden können

Da die verfügbaren Informationen der Dienstbeschreibungen und Annotationen zur Beschreibung von Navigations- und Datenflüssen nicht ausreichend sind, um die im Anwendungsszenario beschriebene Anwendung komplett umzusetzen, sollen weitere geeignete Dienstannotationen erstellt werden, um die Anwendungs- und Benutzungsschnittstellenbeschreibung mit notwendigen Informationen zu ergänzen. Für die Erstellung der zu realisierenden Dienstannotationen soll das im Rahmen des ServFace-Projektes[6] entwickelte Annotationen-Modell „ServFace Annotation Model“ erweitert werden. Bestehende Elemente des Annotationen-Modells sollen soweit möglich für die Umsetzung der zu realisierenden Dienstannotationen einbezogen werden.

/AD3: Unterstützung verschiedener Zielplattformen

Die mit dem Softwarewerkzeug zu erstellenden Anwendungen sollen für die Weiterverarbeitung im Generierungsverfahren beliebige Zielplattformen unterstützen. Um die individuellen Eigenschaften verschiedener Zielplattformen berücksichtigen zu können, müssen die Dienstannotationen eine Bindung an Plattformen unterstützen.

/AD4: Unterstützung erweiterter Kontextbeschreibungen

Um die Gültigkeit von Dienstannotationen an unterschiedliche Kontextinformationen wie beispielsweise die Systemsprache oder den Standort zu knüpfen, muss für die zu erstellenden Dienstannotationen eine Bindung an verschiedene Kontexte unterstützt werden.

Übergeordnete Anforderung

AÜ1: Integration in den Gesamtansatz

Da die in dieser Arbeit zu konzipierenden Dienstannotationen und das zu realisierende Softwarewerkzeugs maßgeblich aus dem Gesamtansatz aus [Fel11], welcher in Absatz 1.2 vorgestellt wurde, motiviert sind, stellt die Integration dieser Komponenten in die Prozesskette des Gesamtansatzes eine übergeordnete Anforderung dar.

[6] http://www.servface.eu/

4. Konzept

Auf Basis der identifizierten Anforderungen und den Erkenntnissen aus dem Vergleich bestehender Ansätze werden in Absatz 4.1 zunächst geeignete Dienstannotationen für eine Anwendungs- und Benutzeroberflächenbeschreibung vorgestellt. Daraufhin wird in Absatz 4.2 eine Software zur werkzeuggestützten Erstellung der Dienstannotationen konzipiert.

4.1. Dienstannotationen

Zur Entwicklung der Dienstannotationen wurde das im Kapitel 2.3.1.2 vorgestellte *ServFace Annotation Model* herangezogen und erweitert, da das Modell bereits über einen bewährten Referenzierungsmechanismus zum Adressieren von Dienstelementen verfügt und eine Vielzahl von Dienstannotationen bereitstellt, die zur Entwicklung der erweiterten Dienstannotationen verwendet werden können. In Abbildung 4.1 wird das erweiterte *ServFace Annotation Model* dargestellt.

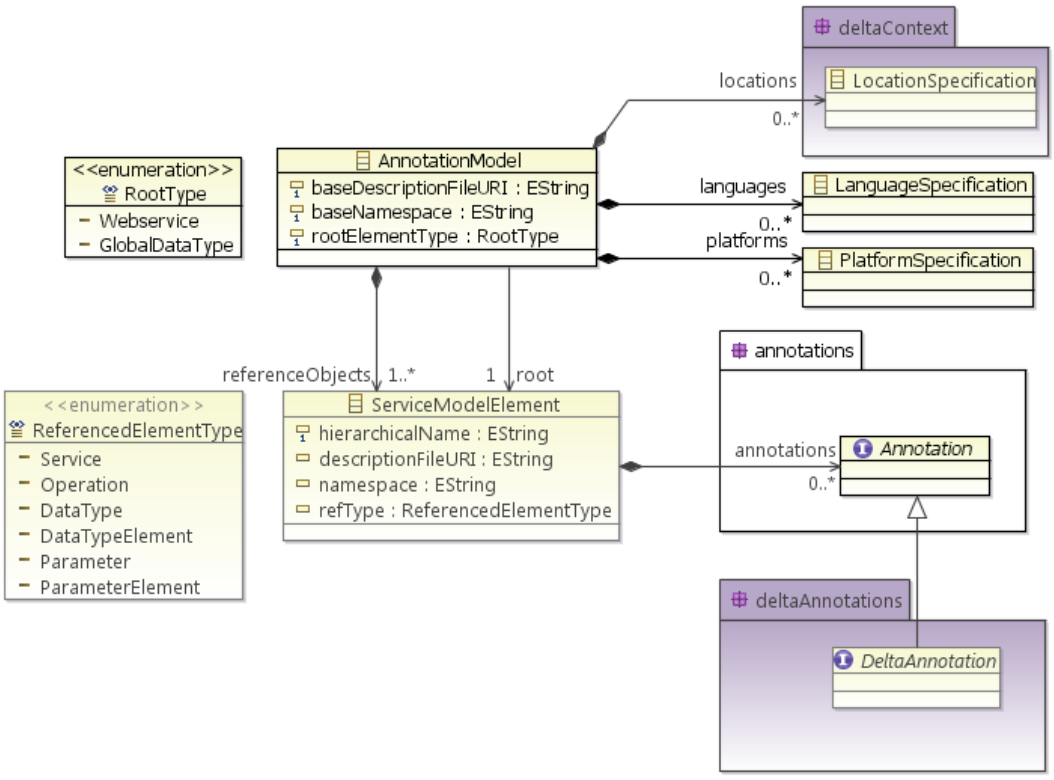


Abbildung 4.1.: Erweitertes *ServFace Annotation Model*

Das Modell wurde durch die Pakete *deltaContext* und *deltaAnnotations* erweitert. Das Paket *deltaContext* ergänzt hierbei den ursprünglichen Beschreibungsumfang von plattformspezifischen und sprachspezifischen Kontextabhängigkeiten um den Kontexttyp *LocationSpecificationen*, welcher zusätzlich die Kontextbindung von Dienstannotationen an Standorte ermöglicht und somit der Anforderung des Anwendungsszenarios (Kapitel 3.1) nachkommt. Hierzu wird ein Kontextelment vom Typ *LocationSpecificationen* durch einen diskreten Bezeichner wie beispielsweise *home* oder *office* gekennzeichnet, um die standortbezogene Einschränkung auszudrücken. Die zur Erweiterung des Modells entwickelten Dienstannotationen besitzen die von *Annotation* abgeleitete, gemeinsame Schnittstelle *DeltaAnnotation* und werden im Paket *deltaAnnotations* gehalten.

Im Folgenden werden die konzipierten Dienstannotationen, in Anlehnung an die Beschreibung der Entwurfsmuster der *„Gang of Four“* [GHJV95], anhand der Eigenschaften *Motivation, Anwendbarkeit, Struktur, Attributwerte* und *Resultat* detailliert beschrieben.

4.1.1. Application

Motivation

Die dynamische Generierung von Benutzerobflächen für Dienste wird mittlerweile durch eine Reihe von unterschiedlichen Ansätzen ermöglicht und bietet dem Nutzer einen direkten Zugriff auf dienstbasierte Funktionalitäten. Gewöhnlich sind derart generierte Benutzeroberflächen auf einfachste Interaktionsmöglichkeiten, die nicht über den Aufruf einer Dienstoperation und der Darstellung des Rückgabeparameters hinausgehen, beschränkt. Die Dienstannotation *Application* bildet die Grundlage zur Beschreibung interaktiver, dienstbasierter Anwendungen, indem ermöglicht wird, die Menge von Dienstoperationen, die vom Nutzer direkt ansteuerbar sind auf eine Teilmenge beschränkt werden.

Anwendbarkeit

Die Dienstannotation *Application* wird zur initialen Beschreibung einer interaktiven Anwendung genau einem Dienst zugeordnet.

Struktur

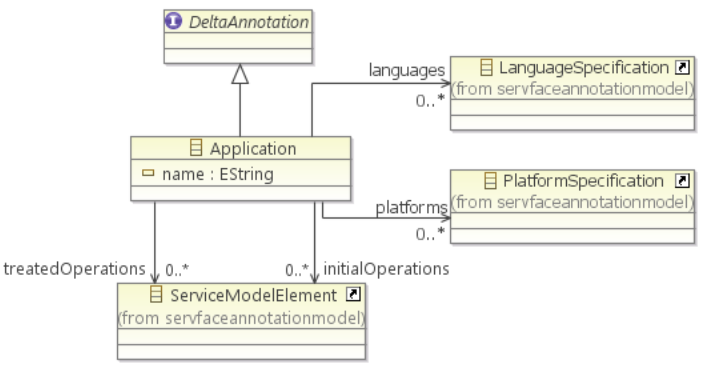


Abbildung 4.2.: Struktur der Dienstannotation *Application*

Attributwerte

- **name**

  - Name der Anwendung

  Der Name wird für die Installation der generierten Anwendung auf dem Zielgerät als Anwendungsname verwendet und kann auch zur Darstellung in der Anwendung genutzt werden. Darüber hinaus wird der Name für den Generierungsprozess als Bestandteil des eindeutigen Bezeichners der Anwendung verwendet.

- **treatedOperations**

  - Liste aller für die Anwendung benötigter Dienstoperationen

  Die in der Liste referenzierten Dienstoperationen, welche auch unterschiedlichen Diensten angehören können, werden jeweils einer Seite der zu generierenden Anwendung zugeordnet.

- **initialOperations**

  - Liste von initialen Dienstoperationen

  Die Dienstoperationen werden in der durch die Liste vorgegebenen Reihenfolge als initiale Navigationsmöglichkeiten im Startzustand der zu generierenden Anwendung dargestellt.

- **languages**

  - Liste von sprachspezifischen Kontextbeschreibungen

  Erlaubt eine unterschiedliche Beschreibung der Dienstannotation für verschiedene Sprachen.

- **platforms**

  - Liste von plattformspezifischen Kontextbeschreibungen

  Die Kontextbindung an Plattformen erlaubt die Anwendung für verschiedene Plattformen unterschiedlich zu beschreiben.

Resultat

Mit Hilfe der Dienstannotation *Application* wird das Grundgerüst einer interaktiven, dienstbasierten Anwendung beschrieben. Für jede Dienstoperation aus der Liste aller in der Anwendung verwendeten Operationen wird eine Zustand der Anwendung abgeleitet, wobei für jede Dienstoperationen, die zusätzlich in der Liste der initialen Operationen geführt wird, ein Navigationspfad vom Startzustand der Anwendung generiert wird.

4.1.2. NavigationChoice

Motivation

Ein wesentlicher Bestandteil zur Beschreibung interaktiver Anwendungen besteht in der Bestimmung einer individuellen Anwendungsstruktur, die durch Navigations- und Datenflüsse gekennzeichnet ist. In Bezug auf dienstbasierte Anwendungen entspricht ein Navigationsfluss einem Übergang zwischen zwei Zuständen einer Anwendung, wobei jeder Zustand einer Dienstoperation zugeordnet ist. Als Ergänzung zu *Application* wird durch die Annotation *NavigationChoice* die Möglichkeit zur Beschreibung einer dienstbasierten Anwendungsstruktur bereitgestellt, womit sich anhand dieser beiden Dienstannotationen bereits interaktive, dienstbasierte Anwendungen ausdrücken lassen.

Anwendbarkeit

Die Annotation kann auf alle Dienstoperationen angewendet werden.

Struktur

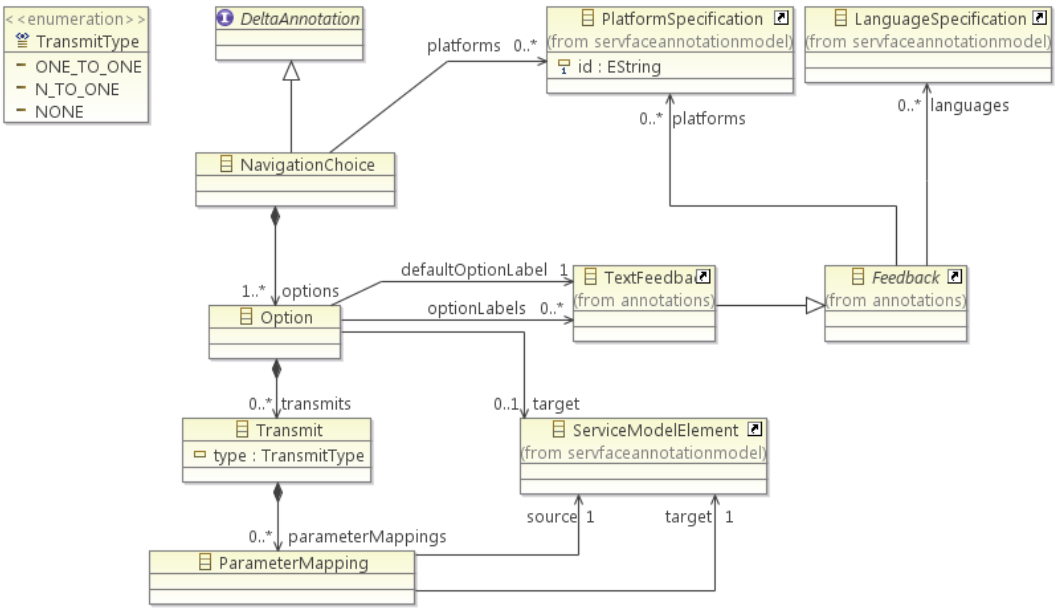


Abbildung 4.3.: Struktur der Dienstannotation *NavigationChoice*

Attributwerte

- **options**

  - Liste von Navigationsoptionen

  Alle in der Liste vorkommenden Optionen werden durch ein Navigationselement zum Navigationaufruf auf der generierten Benutzeroberfläche der Dienstoperation dargestellt.

- **Option → optionLabels**

  - Liste von Bezeichnern für eine Option

  Die Liste ermöglicht die Bereitstellung von sprachspezifischen Bezeichnern, um für ein Navigationselement unterschiedliche Bezeichner für verschiedene Sprachen definieren zu können. Dazu werden die Bezeichner mit der ServFace Dienstannotation *TextFeedback* beschrieben.

- **Option → defaultOptionLabel**

  - Standardbezeichner für eine Option

  Sollte kein sprachspezifischer Bezeichner für den Kontext zutreffen wird *defaultOptionLabel* als Standardbezeichner verwendet.

- **Option → target**

  - Zieloperation der Option

  Der Anwendungszustand, der auf die festgelegte Zieloperation abbildet, wird nach dem Auslösen der Navigation über das Navigationselement aufgerufen. Es können nur Dienstoperationen referenziert werden.

- **Option → transmits**

  - Liste von Datenübertragungen eines Navigationspfades

- **Transmit → type**

  - Typ der Datenübertragung

  Der Typ gibt Auskunft darüber, ob die Übertragung zwischen zwei Parametern (ONE_TO_ONE) oder einem Parameterwert aus einer Liste und einem Parameter (N_TO_ONE) stattfindet. Wenn keine Datenübertragung vorgesehen ist, muss dies durch den Wert NONE gekennzeichnet werden.

- **Transmit → parameterMappings**

  - Liste von Parameterabbildungen

  Parameterabbildungen übertragen den Wert eines Parameters bei einem Navigationsschritt. Dabei wird der übertragene Wert automatisch in das entsprechende Eingabefeld der generierten Benutzeroberfläche des Zielzustands eingetragen.

- **ParameterMapping → source/target**

  - Quell- und Zielparameter der Abbildung für die Beschreibung des Datenflusses innerhalb des Navigationsflusses.

- **platforms**

  - Liste von plattformspezifischen Kontextbeschreibungen

  Die Kontextbindung an Plattformen ermöglicht es unterschiedliche Navigations- und Datenflüssen für verschiedene Plattformen zu beschreiben.

Resultat

Durch die Dienstannotation werden auf der Anwendungsseite der annotierten Dienstoperation Navigationselemente dargestellt, die das Auslösen von Zustandsübergängen zu einem nächsten Anwendungszustand ermöglichen. Über die optionale Beschreibung von Parameterabbildungen können zusätzlich Datenflüsse definiert werden, die zu einer automatischen Dateneingabe von Parameterwerten bei generierten Eingabefeldern führen.

4.1.3. ResultDependentNavigation

Motivation

Feststehende Navigations- und Datenflüsse einer Anwendung können mit der Dienstannotation *NavigationChoice* vollständig beschrieben werden. Es besteht allerdings nicht die Möglichkeit bedingte Navigationsflüsse zu beschreiben. Die Dienstannotation *ResultDependentNavigation* ermöglicht das Definieren von Navigationsflüssen in Abhängigkeit der Erfüllung bestimmter Ausdrücke.

Anwendbarkeit

Die Dienstannotation kann auf alle Dienstoperationen, die eine bedingte Navigation zu einem nachfolgenden Zustand der Anwendung beschreiben sollen, angewendet werden.

Struktur

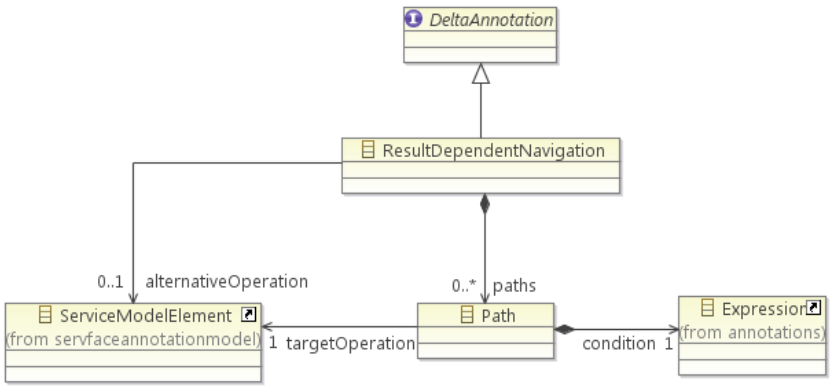


Abbildung 4.4.: Struktur der Dienstannotation *ResultDependentNavigation*

Attributwerte

- **paths**

  - Liste der bedingten Navigationspfade

  Sollten die Ausdrücke mehrerer bedingter Navigationspfade erfüllt sein, so wird stets die Navigation der ersten Übereinstimmung aus der Liste angewendet.

- **Path → condition**

  - Ausdruck

  Der Ausdruck wird mit Hilfe des ServFace Beschreibungselements Expression formuliert. Dabei stehen drei konditionale Beschreibungssprachen zur Verfügung, die gewöhnlich zur Auswertung des Rückgabeparameters der Dienstoperation genutzt werden.

- **Path → targetOperation**

  - Zieloperation des Navigationspfades

- **alternativeOperation**

  - alternativer Navigationspfad

  Sollte kein Ausdruck der bedingten Navigationspfade zutreffen, wird der alternative Navigationspfad angewendet.

Resultat

Die Anwendung der Dienstannotation führt zu einem bedingten Zustandsübergang zwischen zwei Zuständen der Anwendung, wobei die Bedingung üblicherweise an den Rückgabeparameter des Dienstaufrufes gebunden ist.

4.1.4. DisplayAtomicInOut

Motivation

Der Aufruf von Dienstoperationen über generierte Benutzerschnittstellen wird gewöhnlich durch die Request-Response Interaktion mit Diensten beeinflusst. Für die Eingabe und Ausgabe eines Dienstaufrufes werden unterschiedliche Seiten generiert, welche separat dargestellt werden. Dies führt vor allem bei Dienstoperationen, die häufig hintereinander ausgeführt werden, wie beispielsweise Suchoperationen, zu einer verschlechterten Benutzbarkeit, da der Nutzer für jede weitere Suchanfrage erneut zur Eingabeseite navigieren muss. Die Dienstannotation *DisplayAtomicInOut* ermöglicht eine gemeinsame Darstellung von Eingabe und Ausgabe eines Operationsaufrufes auf einer Seite, wie im Anwendungsszenario (Kapitel 3.1) für die Patientensuche vorgesehen.

Anwendbarkeit

Die Dienstannotation kann auf alle Dienstoperationen, die Eingabe- und Ausgabeparameter besitzen, angewendet werden.

Struktur

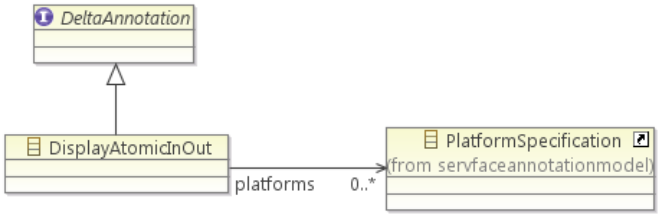


Abbildung 4.5.: Struktur der Dienstannotation *DisplayAtomicInOut*

Attributwerte

- **platforms**

  - Liste von plattformspezifischen Kontextbeschreibungen

  Die Kontextbindung an Plattformen erlaubt die Anwendung der Dienstannotation auf festgelegte Plattformen zu beschränken.

Resultat

Die zur Eingabe des Operationsaufrufes generierte Benutzeroberfläche und die Rückgabe des Dienstaufrufes werden auf der selben Seite dargestellt. Hierzu wird die Ansicht der Seite in zwei inhaltlich unabhängige Bereiche unterteilt. Während der erste Bereich die Benutzungsoberfläche zur Eingabe und zum Dienstaufruf darstellt, wird in dem zweiten Bereich der Rückgabewert des Dienstaufrufes visualisiert.

4.1.5. HumanReadableString

Motivation

Neben der Generierung von Eingabeoberflächen zum Dienstaufruf gehört die nutzerfreundliche Darstellung des Rückgabewertes eines Dienstaufrufes zu den wichtigsten Aufgaben der Inferenzkomponente. Für nicht-textbasierte Rückgabewerte, die beispielsweise in Form eines Bitstreams Bilder oder Videos repräsentieren, bietet die ServFace-Dienstannotation *MIME-Type* bereits entsprechende Auszeichnungsmöglichkeiten, die zur korrekten Darstellung des Medien-Typs beitragen.

Keine Einflussmöglichkeit hingegen gibt es bei der generierten Darstellung von komplexen Datentypen. Während die komplette Darstellung eines komplexen Datentypen vor allem in Form einer Liste sehr unübersichtlich werden kann, besteht bei einer automatisiert vereinfachten Darstellung eines komplexen Datentypen die Gefahr wichtige Informationen zu verbergen. Mit Hilfe der Dienstannotation *HumanReadableString* wird eine individuelle

Darstellungsbeschreibung für komplexe Datentypen bereitgestellt, um eine nutzerfreundliche Generierung von Rückgabewerten eines Dienstaufrufes zu ermöglichen.

Anwendbarkeit

Die Dienstannotation kann nur auf komplexe Rückgabeparameter von Dienstoperationen angewendet werden.

Struktur

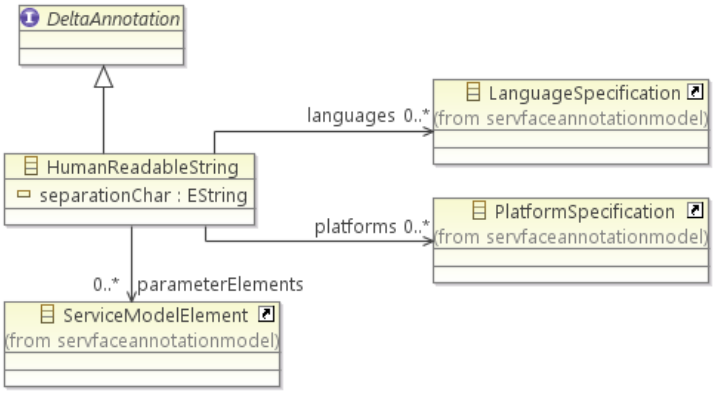


Abbildung 4.6.: Struktur der Dienstannotation *HumanReadableString*

Attributwerte

- **separationChar**

  - Trennungszeichen

  Das Trennungszeichen wird zwischen den darzustellenden Elementen des komplexen Datentypen ausgegeben, um eine individuelle, optische Abgrenzung zwischen den einzelnen Elementen beispielsweise durch ein Komma oder Semikolon zu ermöglichen.

- **parameterElements**

  - Liste der darzustellenden Parameterelemente

  Es können nur primitive Kindelemente eines komplexen Rückgabeparameters referenziert werden. Im Fall einer Verschachtelung mehrerer komplexer Datentypen können jeweils auch die primitiven Kindelemente der komplexen Kindelemente referenziert werden. Die Darstellungsreihenfolge der Elemente wird durch die Reihenfolge der Referenzen in der Liste bestimmt.

- **languages**

  - Liste von sprachspezifischen Kontextbeschreibungen

  Die Kontextbindung an Sprachen soll individuelle Darstellungsbeschreibungen in Bezug auf sprachtypische bzw. landesübliche Formatierungseigenschaften ermöglichen.

- **platforms**

  - Liste von plattformspezifischen Kontextbeschreibungen

  Die Kontextbindung an Plattformen ermöglicht die individuelle Darstellung von komplexen Rückgabewerten unter Berücksichtigung von plattformspezifischen Eigenschaften, wie beispielsweise der Größe des zur Verfügung stehenden Darstellungsbereiches.

Resultat

Die Dienstannotation *HumanReadableString* ermöglicht eine Darstellungsbeschreibung für Rückgabeparameter, die durch einen komplexen Datentyp beschrieben sind. Dabei kann durch die Dienstannotation beschrieben werden, welche primitiven Elemente des komplexen Datentyps in einer festgelegten Reihenfolge und durch ein festgelegtes Trennungszeichen zur Darstellung herangezogen werden.

4.1.6. DisabledParameter

Motivation

Um den Funktionsumfang zu erhöhen, bieten viele Dienstoperationen die Möglichkeit, einen Dienstaufruf durch optionale Eingabeparameter zu konkretisieren. Dadurch haben Dienstoperationen häufig eine Vielzahl von Eingabeparametern, von denen nur wenige für den Dienstaufruf benötigt werden. Da die funktionale Dienstbeschreibung keine Unterscheidung zwischen optionalen und obligatorischen Parametern erlaubt, würde die automatische Generierung einer Benutzerschnittstelle stets alle Eingabeparameter berücksichtigen, was sich negativ auf die Benutzerfreundlichkeit auswirken würde. Mit Hilfe der Dienstannotation *DisabledParameter* sollen einzelne Parameter und Parameterelemente vom Generierungsverfahren ausgeschlossen werden und somit nicht zur Ableitung von Benutzerschnittstellenelementen beitragen. Im Anwendungsszenario (Kapitel 3.1) wird die Dienstannotation genutzt, um beim Bearbeiten von Patientendaten die ID nicht editieren zu können.

Anwendbarkeit

Die Dienstannotation kann auf alle Parameter und Parameterelemente, die sowohl zur Eingabe als auch zur Ausgabe eines Dienstaufrufes dienen, angewendet werden.

Struktur

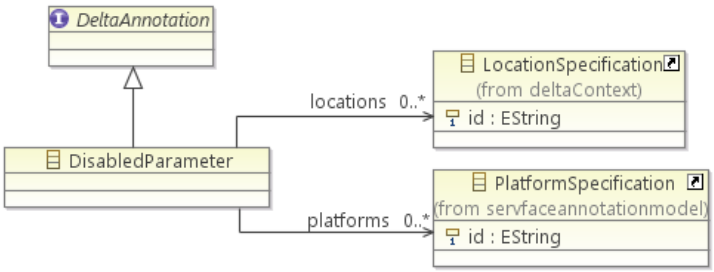


Abbildung 4.7.: Struktur der Dienstannotation *DisabledParameter*

Attributwerte

- **locations**

  - Liste von standortspezifischen Kontextbeschreibungen

  Die Kontextbindung an Standorte ermöglicht das Ausblenden von Parametern bzw. Parameterelementen in Abhängigkeit des gegenwärtigen Standortes.

- **platforms**

  - Liste von plattformspezifischen Kontextbeschreibungen

  Die Kontextbindung an Plattformen ermöglicht das Ausblenden von Parametern bzw. Parameterelementen auf bestimmte Plattform zu beschränken.

Resultat

Parameter bzw. Parameterelemente, die mit der Dienstannotation *DisabledParameter* ausgezeichnet sind, werden bei der Benutzerschnittstellengenerierung nicht berücksichtigt. Demnach werden für diese Elemente keine Bezeichner und Eingabefelder erzeugt. Wird ein Element eines Rückgabeparameters mit dieser Dienstannotation ausgezeichnet, so wird der entsprechende Bestandteil der Rückgabe ausgeblendet.

4.1.7. PlatformProvidedEntity

Motivation

Eine Möglichkeit, die Benutzbarkeit von generierten Benutzerschnittstellen zu verbessern, besteht in der automatisierten Bereitstellung von Eingabewerten, die mit Hilfe spezieller Komponenten auf dem Zielgerät abgerufen werden können (vgl. [Fel11], Kapitel 4.2.5). Vor allem mobile Endgeräte ermöglichen den Zugriff auf eine Vielzahl von Systemparametern und Kontextinformationen. Die Dienstannotation *PlatformProvidedEntity* ermöglicht es, diese Informationen als Eingabeparameter für Dienstaufrufe zu nutzen und dem Benutzer auf diese Weise den Dienstaufruf zu vereinfachen. Im Anwendungsszenario (Kapitel

3.1) soll mit der Dienstannotation die Bereitstellung von GPS-Koordinaten vom mobilen Endgerät realisiert werden.

Anwendbarkeit

Die Dienstannotation kann auf alle Eingabeparameter und Eingabeparameterelemente, die dem Datentyp des von der Plattform zur Verfügung gestellten Parameters entsprechen, angewendet werden.

Struktur

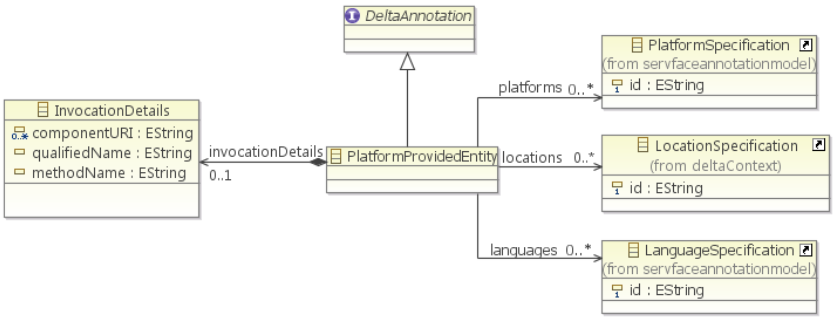


Abbildung 4.8.: Struktur der Dienstannotation *PlatformProvidedEntity*

Attributwerte

- **invocationDetails**

  - Aufrufinformationen eines von der Plattform zur Verfügung gestellten Parameters

- **InvocationDetails → componentURI**

  - Identifikator der Komponente

  Die Komponente stellt der zu generierenden Anwendung die Funktionalität zum Abruf des Parameters vom Zielgerät bereit.

- **InvocationDetails → qualifiedName**

  - eindeutiger Bezeichner des von der Plattform bereitgestellten Parameters

- **InvocationDetails → methodName**

  - Name der aufzurufenden Methode

  Der Methodenname dient als Schnittstelle, um den Abruf des von der Plattform bereitgestellten Parameters über die von der Komponente zur Verfügung gestellten Funktionalität auszulösen. Es werden nur parameterlose Methodenaufrufe unterstützt

- **locations**

  - Liste von standortspezifischen Kontextbeschreibungen

  Die Kontextbindung an Standorte ermöglicht die Bereitstellung des Parameters vom Zielgerät auf festgelegte Standorte zu beschränken.

- **platforms**

  - Liste von plattformspezifischen Kontextbeschreibungen

  Die Kontextbindung an Plattformen ermöglicht die Bereitstellung des Parameters vom Zielgerät auf bestimmte Plattformen zu beschränken.

Resultat

Wird ein Eingabeparameter mit der Dienstannotation *PlatformProvidedEntity* ausgezeichnet, so werden für diesen Parameter keine Benutzerschnittstellenelemente generiert. Stattdessen wird der Parameterwert automatisch vom ausführenden System abgerufen und für den Aufruf entsprechender Dienstoperationen verwendet.

4.1.8. CustomWidget

Motivation

Dienstannotationen ermöglichen eine abstrakte und plattformunabhängige Benutzerschnittstellenbeschreibung für Dienste, um die Generierung von Benutzeroberflächen oder dienstbasierter, interaktiver Anwendungen für beliebige Zielplattformen zu ermöglichen. Es besteht aber noch keine Möglichkeit für die Ein- beziehungsweise Ausgabe spezieller Datentypen angepasste Widgets zu verwenden. Die Dienstannotation *CustomWidget* soll die Anwendung von plattformspezifischen Benutzerschnittstellenelementen und vollständig benutzerdefinierten Bedienelementen ermöglichen.

Anwendbarkeit

Die Dienstannotation kann auf alle Parameter und Parameterelemente, die sowohl zur Eingabe als auch zur Ausgabe eines Dienstaufrufes dienen, angewendet werden.

Struktur

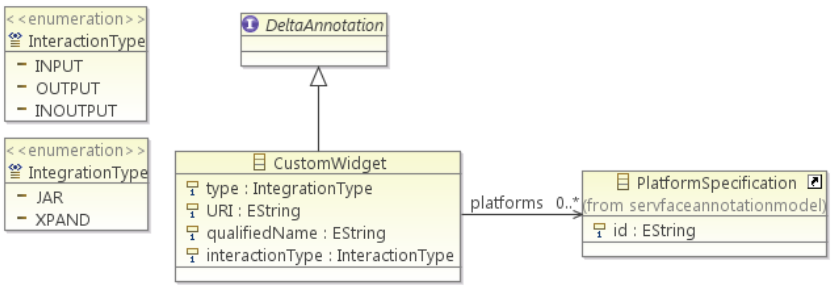


Abbildung 4.9.: Struktur der Dienstannotation *CustomWidget*

Attributwerte

- **type**

  - Typ der Integration

  Das Attribut *type* gibt an, in welcher Form das Bedienelement als Ressource veröffentlicht wurde. Das benutzerdefinierte Bedienelement kann durch ausführbare Programmkomponenten in Form von JAR-Dateien oder Xpand-Skipten zur Verfügung gestellt werden. Aber auch eine Erweiterung zur Einbindung weiterer Komponenten, wie beispielsweise den User Interface Services, die im Rahmen des CRUISe-Projektes [LMS10] entstanden sind und den Zugriff auf wiederverwendbare, dienstbasierte Benutzerschnittstellenelemente ermöglichen, ist mit Hilfe der Dienstannotation realisierbar (siehe auch [FHLS10]).

- **URI**

  - Identifikator der veröffentlichten Ressource

- **qualifiedName**

  - eindeutiger Bezeichner des benutzerdefinierten Bedienelements

- **interactionType**

  - Typ der Interaktion

  Als wichtige Information für den Generierungsprozess gibt der *interactionType* an, ob das Bedienelement zur Eingabe beiträgt, die Ausgabe darstellt, oder sowohl für die Eingabe als auch Ausgabe eines Operationsaufrufes genutzt wird.

- **platforms**

  - Liste von plattformspezifischen Kontextbeschreibungen

  Benutzerdefinierte Interaktoren werden in der Regel plattformspezifisch entwickelt und können demnach nur für die Generierung von Zielanwendungen für einzelne Plattformen angewendet werden. Über die Kontextbindung kann diese Beschränkung beschrieben werden.

Resultat

Die Auszeichnung mit der Dienstannotation *CustomWidget* definiert für einen Parameter ein benutzerdefiniertes Bedienelement. Im Generierungsverfahren wird der Parameter vom Inferenz-Mechanismus ausgeschlossen und stattdessen durch das benutzerdefinierte Bedienelement repräsentiert.

4.1.9. ResultDependentString

Motivation

Da die Semantik der Rückgabewerte von Operationsaufrufen für Menschen häufig unverständlich ist, leidet die Benutzbarkeit bei einer unverarbeiteten Darstellung von Rückgabewerten. Die Dienstannotation *ResultDependentString* soll das Ersetzen eines Rückgabewertes in einen geeigneten Textbezeichner ermöglichen. Auf diese Weise kann zum Beispiel ein boolescher Rückgabewert, der eine Aussage über den Erfolg der ausgeführten Dienstoperation trifft, diesen Erfolg in einer menschenverständlichen Form beschreiben.

Anwendbarkeit

Die Dienstannotation kann auf Rückgabeparameter von Dienstoperationen angewendet werden, um die Rückgabe des Operationsaufrufes durch eine benutzerdefinierte Bezeichnung zu ersetzen.

Struktur

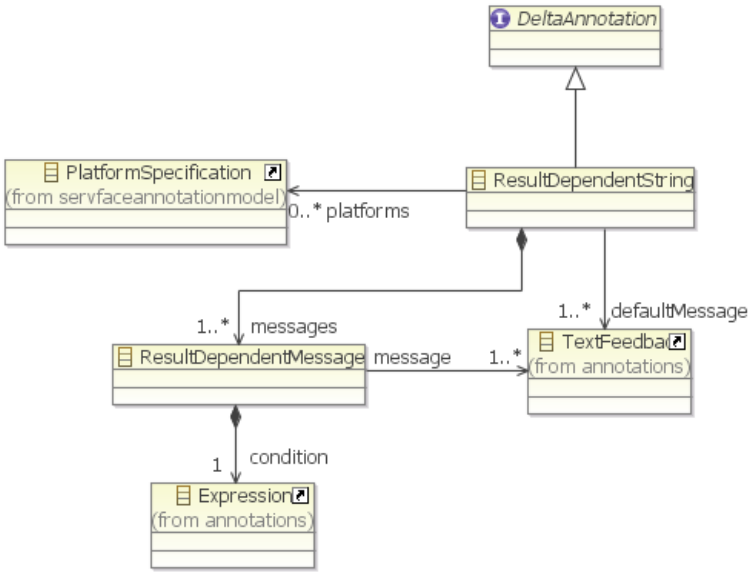


Abbildung 4.10.: Struktur der Dienstannotation *ResultDependentString*

Attributwerte

- **messages**

  - Liste von Ersetzungsbezeichnern

  Über diese Liste werden die Textbezeichner, welche als Ersetzung der Rückgabewerte genutzt werden, definiert. Hierzu wird die ServFace Dienstannotation *TextFeedback* verwendet, um zugleich die Beschreibung von unterschiedlichen Bezeichnern für verschiedene Sprachen festlegen zu können. Es können belebig viele Ersetzungsbezeichner definiert werden, die jeweils an einen Ausdruck gebunden sind.

- **defaultMessage**

  - alternativer Bezeichner

  Sollte kein Ausdruck zutreffen, wird der Rückgabewert mit dem Textbezeichner dieses Parameters ersetzt.

- **expression**

  - Ausdruck

  Der Ausdruck wird mit Hilfe des ServFace Beschreibungselements *Expression* formuliert. Dabei stehen drei konditionale Beschreibungssprachen zur Verfügung, die im Kontext gewöhnlich zur Auswertung des Rückgabeparameters der Dienstoperation genutzt werden.

- **platforms**

  - Liste von plattformspezifischen Kontextbeschreibungen

  Die Kontextbindung an Plattformen ermöglicht die Ersetzung des Rückgabewertes auf bestimmte Plattformen zu beschränken.

Resultat

Eine Anwendung dieser Dienstannotation führt dazu, dass der Rückgabewert eines Dienstaufrufes nicht dargestellt wird. Stattdessen wird in Abhängigkeit eines auszuwertenden Ausdrucks ein benutzerdefinierter Textbezeichner dargestellt, wodurch die Benutzbarkeit der generierten Benutzeroberfläche verbessert werden soll.

4.1.10. SessionToken

Motivation

Viele Dienste bieten Zugriff auf sicherheitsrelevante Funktionaltäten, die nur authentifizierten Nutzern vorbehalten sein sollen. Die ServFace Dienstannotation *Authentication* bietet bereits die Möglichkeit Dienste oder Dienstoperationen durch unterschiedliche Authentisierungsmechanismen zu sichern. Um eine sitzungsbasierte Authentisierung mit Sitzungsschlüssel für Dienstaufrufe zu vereinfachen, wurde in Ergänzung die Dienstannotation *SessionToken* entwickelt.

Anwendbarkeit

Die Dienstannotation kann auf alle Parameter angewendet werden, die einen Sitzungsschlüssel zur Authentifizierung darstellen.

Struktur

Abbildung 4.11.: Struktur der Dienstannotation *SessionToken*

Attributwerte

- **providedByHost**

  Der boolesche Wert des Parameters gibt Auskunft darüber, ob der Sitzungsschlüssel von der Hostanwendung zur Verfügung gestellt wird.

- **authenticationOperation**

  - Authentisierungsoperation

  Die Authentisierungsoperation bietet dem Nutzer die Möglichkeit sich zu Authentifizieren. Die Operation wird benötigt, wenn eine geschützte Dienstoperation aufgerufen wurde und der Sitzungsschlüssel ungültig ist. Als Rückgabewert liefert die Authentisierungsoperation den Sitzungsschlüssel nach einem erfolgreichen Login.

Resultat

Die Anwendung der Dienstannotation führt dazu, dass für den annotierten Parameter keine Benutzerschnittstellenelemente generiert werden. Stattdessen wird dem Parameter bei einem Dienstaufruf der Sitzungsschlüssel zugeordnet. Sollte der Sitzungsschlüssel nicht gültig sein, so wird der Dienstaufruf unterbrochen und zunächst die Authentisierungsoperation dargestellt, um dem Nutzer die Möglichkeit zu geben, sich zu authentifizieren und den gewünschten Dienstaufruf fortzusetzen.

4.1.11. CallOnInputVisualisation

Motivation

Um die Generierung einer Eingabeoberfläche für Dienstoperation nicht auf die statischen Informationen, die bei Anwendung einer Inferenz aus den funktionalen Dienstbeschreibung gewonnen werden können, zu beschränken, bietet die Dienstannotation *CallOnInputVisualisation* die Möglichkeit, dynamisch über einen zusätzlichen Dienstaufruf zur Ausführungszeit Informationen abzurufen und für die Generierung der Benutzerschnittstelle zu verwenden. Auf diese Weise können beispielsweise Statusinformationen zu Systemzuständen eingebunden werden.

Anwendbarkeit

Die Dienstannotation kann auf alle Dienstoperationen angewendet werden.

Struktur

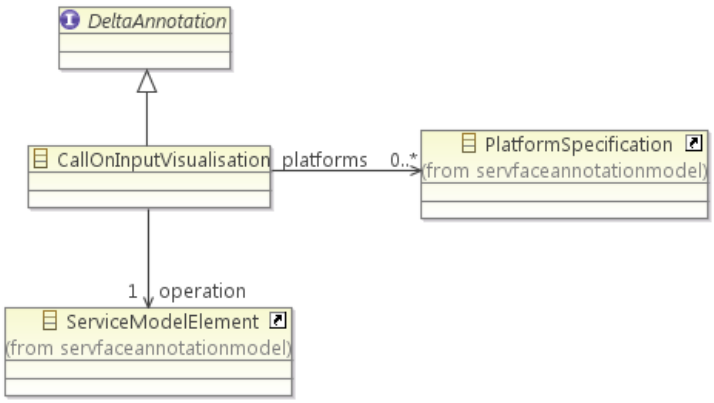


Abbildung 4.12.: Struktur der Dienstannotation *CallOnInputVisualisation*

Attributwerte

- **operation**

  - aufzurufende Dienstoperation

  Bei der aufzurufenden Dienstoperation muss es sich um eine Operation handeln, die ohne Nutzereingaben ausgeführt werden kann.

- **platforms**

  - Liste von plattformspezifischen Kontextbeschreibungen

  Die Kontextbindung an Plattformen ermöglicht den zusätzlichen Dienstaufruf auf bestimmte Plattformen zu beschränken.

Resultat

Eine Anwendung der Dienstannotation führt zur Darstellung des Rückgabewertes der angegebenen Dienstoperation. Der Rückgabewert wird in räumlicher Nähe zur generierten Benutzeroberfläche der Eingabeoperation dargestellt.

4.1.12. Zusammenfassung

Für die Beschreibung der Dienstannotationen wurde das *ServFace Annotation Model* herangezogen und erweitert, um den bestehenden Beschreibungsumfang zusätzlich nutzen zu können. Neben den Dienstannotationen *Application*, *NavigationChoice* und *ResultDependentNavigation*, welche die Beschreibung einer Anwendungsstruktur mit Navigations- und Datenflüssen ermöglichen, wurden eine Vielzahl weiterer Dienstannotationen entworfen,

um sämtliche Funktionalitäten des Anwendungsszenarios aus Kapitel 3.1 und der Anwendungsszenarien aus [Fel11] und [Ber10] beschreiben zu können.

4.2. Graphischer Editor

Die folgenden Abschnitte beschreiben das Konzept eines graphischen Editors zur Instanziierung der im vorigen Abschnitt vorgestellten Dienstannotationen.

4.2.1. Basisarchitektur

Abbildung 4.13 gibt einen Überblick über die Hauptkomponenten des Editors und verdeutlicht anhand des dargestellten Informationsflusses die Abhängigkeiten zwischen den Komponenten.

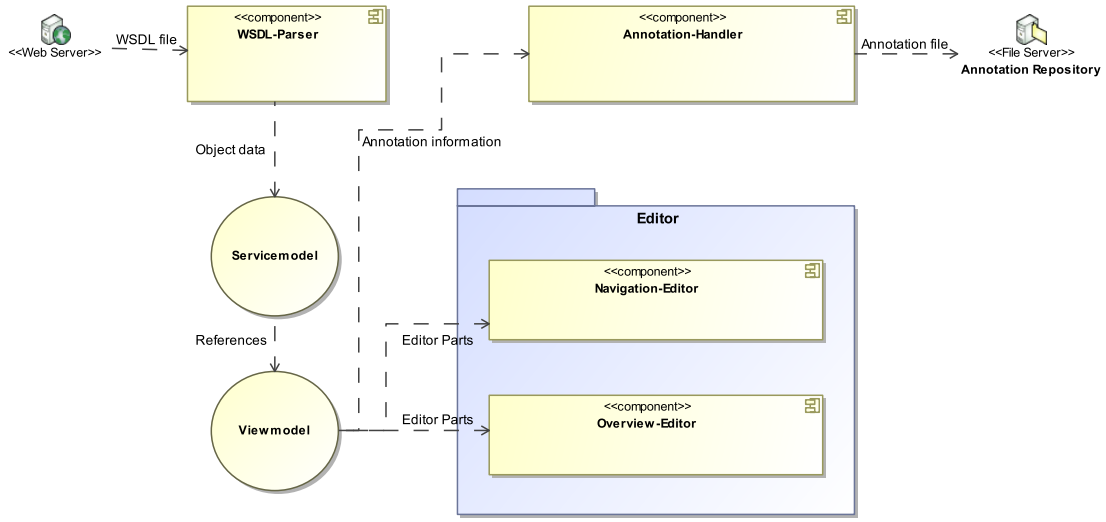


Abbildung 4.13.: Überblick über die Basisarchitektur

Der Import einer WSDL-Datei stellt den Ausgangspunkt bei der Arbeit mit dem Editor dar. Somit ist schon das Anlegen einer neuen Anwendung mit der Angabe einer URI zu einer WSDL-Dienstbeschreibung verbunden. Mit Hilfe des *WSDL-Parsers* werden benötigte Informationen der funktionalen Dienstbeschreibung in das *Servicemodel* übertragen, um während der Bearbeitung zentral zur Verfügung zu stehen. Da das *Servicemodel* aufgrund seiner Struktur und Integrität der Daten nicht für eine direkte Bearbeitung in den graphischen Editoransichten verwendet werden kann, wurde das *Viewmodel* abgeleitet. Das *Viewmodel* bestimmt die Struktur der editierbaren Elemente in den Editoransichten und enthält sämtliche Informationen, um die entwickelten Dienstannotationen aus Abschnitt 4.1 zu beschreiben. Zur Erstellung einer Anwendung werden zwei unterschiedliche Editoransichten realisiert. Der *Navigation-Editor* unterstützt durch verschiedene Editorwerkzeuge vor allem die Erstellung von Navigationsübergängen und Parameterabbildungen zwischen Anwendungszuständen. Um zu jeder Zeit einen Überblick über die entworfene Anwendung

zu erhalten, bietet der *Overview-Editor* eine übersichtliche Darstellung aller Anwendungszustände und deren Transitionen. Zur Veröffentlichung einer modellierten Anwendung dient der *Annotation-Handler*. Die Komponente leitet aus den editierten Elementen des *Viewmodels* Dienstannotationen ab und serialisiert diese im XMI-Format[7].

4.2.2. Konzeption der Oberfläche

Die Oberfläche des Editors soll in die drei Bereiche Serviceansicht, Eigenschaftenbereich und Editorbereich eingeteilt werden. Der Bereich, welcher in Abbildung 4.14 mit *Service View* gekennzeichnet, ist stellt die Serviceansicht dar. Er bietet eine baumartig strukturierte Übersicht über die importierten Dienste und allen relevanten Dienstelementen wie beispielsweise den Dienstoperationen. Auf diese Weise erhält der Nutzer stets einen Überblick über die zur Verfügung stehenden Dienstfunktionalitäten und kann in dieser Ansicht für einzelne Dienstoperationen eine neue Anwendungsseite festlegen, welche daraufhin im Editorbereich weiter editiert werden kann. Der in Abbildung 4.14 mit *Properties* gekennzeichnete Abschnitt im unteren Bereich der Anwendungsoberfläche dient der Darstellung und dem Editieren von Eigenschaftswerten. Hierbei werden Eigenschaftswerte von Objekten aus dem *Service View* und den Editoransichten unterstützt. Zur Darstellung der Eigenschaftswerte genügt eine Selektion des gewünschten Objektes in der jeweiligen Ansicht.

Den bedeutendsten Teil der Anwendungsoberfläche stellt der Editorbereich dar. Dieser Bereich wird für die Ausführung der beiden graphischen Editoren, welche im Folgenden detailliert beschrieben werden, genutzt:

Navigation-Editor

Der *Navigation-Editor* stellt die Haupteditoransicht der Anwendung zur Verfügung und soll das Editieren sämtlicher anwendungsrelevanter Dienstannotationen ermöglichen. Hierzu stellt der Editor für eine anwenderfreundliche Nutzung mehrere Werkzeuge bereit. Abbildung 4.14 zeigt eine Konzeptzeichnung der gesamten Anwendung mit aktiver Ansicht des *Navigation-Editors*.

Der Editorbereich macht den Hauptbestandteil der Anwendungsoberfläche aus. Während der Bearbeitung sind grundsätzlich beide Editoren geöffnet und der Nutzer hat die Möglichkeit jederzeit zwischen der Editoransicht des *Navigation-Editors* und des *Overview-Editors* zu wechseln. Die Darstellung der graphischen Objekte eines Editors werden aus der Struktur des in Abschnitt 4.2.3 vorgestellten *Viewmodels* abgeleitet. Demnach gliedert sich die Ansicht des *Navigation-Editors* in einen linken und einen rechten Teilbereich, welche jeweils zur Darstellung von Anwendungsseiten dienen. Über die Dienstansicht können für Dienstoperationen Anwendungsseiten angelegt und zur Darstellung im linken beziehungsweise rechten Teilbereich des Editors festgelegt werden. Auf diese Weise können jeweils zwei Anwendungsseiten gegenübergestellt werden. Da eine Anwendungsseite immer genau

[7] http://www.omg.org/spec/XMI/2.1.1/

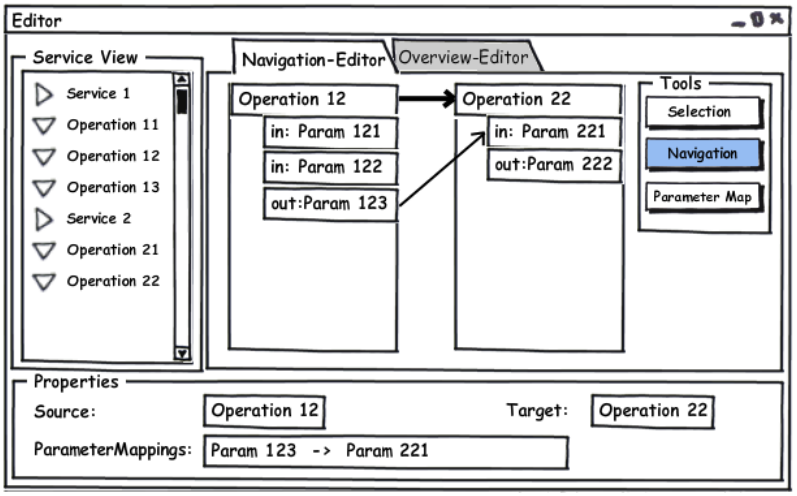


Abbildung 4.14.: Editorkonzept für die Navigationsansicht

einer Dienstoperation zugeordnet ist, wird diese durch den Operationsnamen und den zur Operation gehörenden Ein- und Ausgabeparameter repräsentiert. Navigationsübergänge zwischen Anwendungsseiten und Parameterabbildungen lassen sich so mit Hilfe von entsprechenden Werkzeugen einfach umsetzen.

Werkzeuge werden von der Werkzeugpalette, welche in der Abbildung mit *Tools* gekennzeichnet ist und zur Editoransicht gehört, zur Verfügung gestellt. Sie sollen ein anwenderfreundliches Editieren von Dienstannotationen zur Beschreibung einer dienstbasierten Anwendung ermöglichen. Der *Navigation-Editor* setzt hierzu die folgenden Werkzeuge um:

- **Selection-Tool:** Das Auswahlwerkzeug ist das Standardwerkzeug und dient der Selektion von Editorobjekten. Wenn das Werkzeug aktiviert ist, können sämtliche Editorobjekte mit einem Mausklick ausgewählt werden. Durch eine Selektion wird das jeweilige Editorobjekt durch einen Auswahlrahmen hervorgehoben und die Eigenschaften des Objektes werden in der Eigenschaftenansicht zum Editieren dargestellt.
- **NavigationChoice-Tool:** Das NavigationChoice-Tool ist ein objekterzeugendes Werkzeug. Es dient dem Erstellen eines Navigationsauswahlobjektes, welches dem Element *NavigationChoice* des Viewmodels entspricht. Wenn das Werkzeug aktiv ist, genügt ein Mausklick auf einem Editorteilbereich, um der Anwendungsseite ein Navigationsauswahlobjekt hinzuzufügen. Über das erzeugte Objekt können Navigationsoptionen erstellt werden, die dann als Ausgangspunkt für Navigationsübergänge genutzt werden.
- **NavigationConnection-Tool:** Um Navigationsübergänge von Navigationsoptionen zu Anwendungsseiten erstellen zu können, bietet dieses Verbindungswerkzeug die Möglichkeit, eine Verknüpfung zwischen diesen Objekten zu erstellen. Eine Objektverbindung resultiert in einer gerichteten Verbindungslinie, die vom Editor als Viewmodel-Element *NavigationPathConnection* dargestellt wird. Eine Verknüpfung muss immer von der Quelle (Navigationsoption) im linken Editorteilbereich zum Ziel (Anwendungsseite) im rechten Editorteilbereich aufgespannt werden.

- **ParameterMapping-Tool:** Das ParameterMapping-Tool ist ein Verbindungswerkzeug, das zur Modellierung von Parameterabbildungen genutzt wird. Das Werkzeug kann erst nach der Selektion eines erstellten Navigationsübergangs mit dem Selection-Tool aktiviert werden, um die Zugehörigkeit von den zu erstellenden Parameterabbildungen zu einem Navigationsübergang zu erfassen. Eine Parameterabbildung kann dann von einem Parameter des linken Editorteilbereichs zu einem Parameter, der im rechten Editorteilbereich dargestellt ist und auf den selben Datentypen abbildet, erstellt werden. Das Resultat einer Parameterabbildung wird ähnlich wie ein Navigationsübergang durch eine gerichtete Verbindungslinie dargestellt. Eine Parameterabbildung wird vom Editor als Viewmodel-Element *ParameterMappingConnection* verwaltet.

- **DisableParameter-Tool:** Das Werkzeug ermöglicht den booleschen Eigenschaftswert *isEnabled* von Parametern und Parameterelementen zu wechseln, was bei der Anwendungserstellung zur Anwendung der Dienstannotation *DisabledParameter* (4.1.6) führt und den Parameter somit von der Benutzerschnittstellengenerierung ausschließt. Die Anwendung des Werkzeugs erfolgt durch einen Mausklick auf einen beliebigen Parameter. Ein Deaktivieren des Parameters wird in der Editoransicht in geeigneter Weise dargestellt. Sollte mit dem Werkzeug ein komplexer Parameter deaktiviert werden, so werden auch alle Kindelemente des Parameters deaktiviert.

Overview-Editor

Der Overview-Editor stellt zu jedem Zeitpunkt der Anwendungsmodellierung einen Überblick über die entworfene Anwendungsstruktur bereit. Die Darstellung der Editoransicht erfolgt im Editorbereich der Anwendungsoberfläche, in welchem ein Wechsel zwischen den Editoransichten des Navigation-Editors und Overview-Editors jederzeit möglich ist. Abbildung 4.15 zeigt eine Kozeptzeichnung der Anwendungsoberfläche mit geöffnetem Overview-Editor.

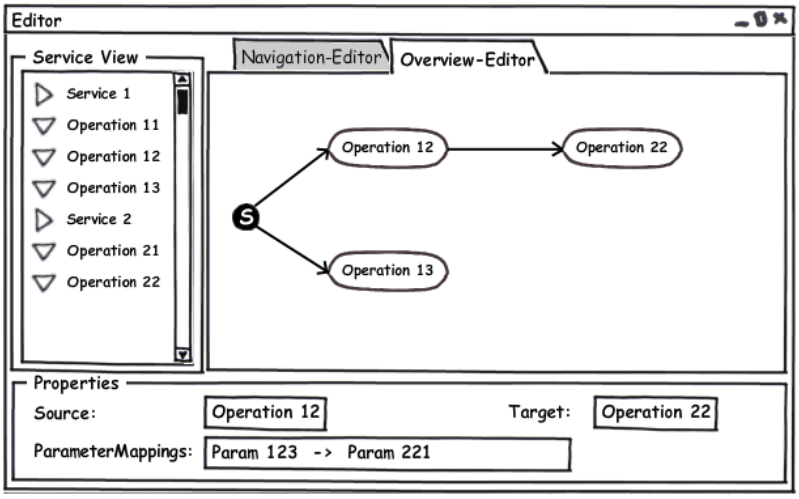


Abbildung 4.15.: Editorkonzept für die Überblickansicht

Zur Darstellung der Anwendungsstruktur wird eine dem Zustandsübergangsdiagramm ähnliche Graphenstruktur angewendet, um die modellierten Navigations- und Datenflüsse abzubilden. Hierbei wird ausgehend von einem Startzustand, von welchem durch Navigationspfade die initialen Anwendungsfunktionalitäten beschrieben werden, automatisch ein vollständiger Graph von allen angelegten Anwendungszuständen erstellt. Die Anwendungszustände, welche als Knoten des Graphen dargestellt sind, werden durch den Namen der mit dem Anwendungszustand verbunden Dienstoperation ausgezeichnet. Sollten an einem Navigationspfad zusätzlich Parameterabbildungen geknüpft sein, so wird dies durch ein entsprechendes Symbol auf dem Navigationspfad gekennzeichnet. Da der Overview-Editor ausschließlich zur Darstellung der Anwendungsstruktur dient, werden dem Nutzer bis auf das Festlegen von initialen Anwendungsseiten keine weiteren Interaktionsmöglichkeiten und somit auch keine Werkzeuge zur Verfügung gestellt.

4.2.3. Konzeption der Modellebene

Für die Datenbeschreibung, der für den Editierprozess benötigten Informationen, werden die beiden Datenmodelle *Servicemodel* und *Viewmodel* eingesetzt.

Servicemodel

Funktionale Dienstbeschreibungen können durch unterschiedliche Beschreibungssprachen formuliert werden. Obwohl für die prototypische Umsetzung des Editors nur die *Web Service Description Language*[Chr01] unterstützt wird, soll das Servicemodel als abstraktes Dienstmodell alle Dienstbeschreibungen abbilden können. Auf diese Weise soll eine mögliche Erweiterung zur Unterstützung weiterer Dienstbeschreibungssprachen zu einen späteren Zeitpunkt ermöglicht werden. Das Servicemodel beinhaltet alle technischen Informationen, die zum Dienstaufruf benötigt werden und beschreibt eine Dienststruktur, welche die annotierbaren Elemente der importierten Dienstbeschreibungen abbildet. Abbildung 4.16 gibt einen Überblick über den Aufbau des Servicemodels.

Die Knoten der funktionalen Schnittstellenbeschreibung werden durch die folgenden Elemente des *Servicemodels* abgebildet:

- **Service:** Ein Dienst stellt die oberste Hierarchieebene einer Dienststruktur dar und gruppiert eine Menge von Operationen, die unter dem vom Dienst beschriebenen Kontext zusammengefasst sind. Das *Servicemodel* kann über das Element *ServiceSet* beliebig viele Dienstbeschreibungen abbilden, wobei Dienstbeschreibungssprachen wie *WSDL* mehrere Dienste innerhalb einer Datei beschreiben können.

- **Operation:** Operationen sind die ausführbaren Konstrukte eines Dienstes. Sie ermöglichen das Abrufen von Informationen bzw. die Manipulation von Daten über entfernte Aufrufe. Operationen sind durch eine Menge von Aufrufparametern und Rückgabeparametern gekennzeichnet.

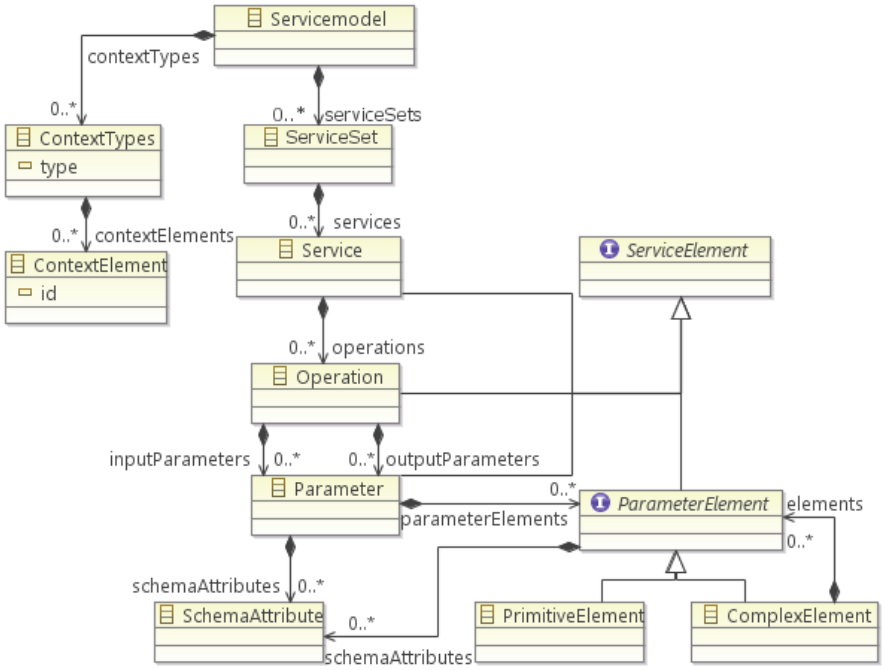


Abbildung 4.16.: Aufbau des Servicemodels

- **Parameter/Parameterelement:** Parameter repräsentieren Daten, die für den Aufruf einer Operation übersendet werden und als Rückgabewert des Dienstaufrufes empfangen werden. Parameter, die keinen primitiven Datentyp beschreiben, werden durch eine Menge von Parameterelementen dargestellt, wobei ein Parameterelement wiederum auf einen primitiven oder komplexen Datentypen abbilden kann. Auf diese Weise kann ein Parameter einen beliebig verschachtelten, komplexen Datentyp beschreiben.

Neben den Dienstelementen beinhaltet das Servicemodel auch Kontexttypen(*ContextType*), um der Anwendung eine frei konfigurierbare Menge von Kontextbeschreibungen zur Verfügung zu stellen. Obwohl zur Beschreibung der Kontextabhängigkeiten für die entwickelten Dienstannotationen des erweiterten *ServFace Annotation Models* nur die drei Kontexttypen *PlatformSpecification*, *LanguageSpecification* und *LocationSpecification* unterstützt werden, soll durch die generische Beschreibung der Kontextinformationen eine Abhängigkeit zum verwendeten Annotationsmodell verhindert werden, um eine Erweiterung beziehungsweise einen Austausch des Annotationsmodells zu ermöglichen. Um auch Schemainformationen der Dienstbeschreibungen abbilden zu können, werden diese in Form von Schemaattributen Parametern und Parameterelementen zugeordnet.

Viewmodel

Das Viewmodel steht zwischen dem Servicemodel und den Editoren. Es ist das zentrale Element der Systemarchitektur und beschreibt eine Abstraktion der Editoransichten. Die Editoransichten werden an die Objekte und Eigenschaften des Viewmodels gebunden, das wiederum über Stellvertreterobjekte auf Daten aus dem Servicemodel zurückgreift. Auf

diese Weise wird die Integrität des Servicemodels gewahrt. Das Viewmodel ermöglicht eine strikte Trennung zwischen der Ansicht und den Daten einer Benutzeroberfläche, wodurch eine bewährte Umsetzung nach dem Model-View-Controller-Entwurfsmuster begünstigt wird. Das Erstellen der Bindungen zwischen den Editoransichten und dem Viewmodel ist einfach, da ein Viewmodel-Objekt als Datenkontext einer Ansicht festgelegt und in geeigneter Weise visualisiert wird. Wenn sich Eigenschaftswerte im ViewModel ändern, werden diese neuen Werte automatisch über die Datenbindung an die Ansichten weitergeleitet.

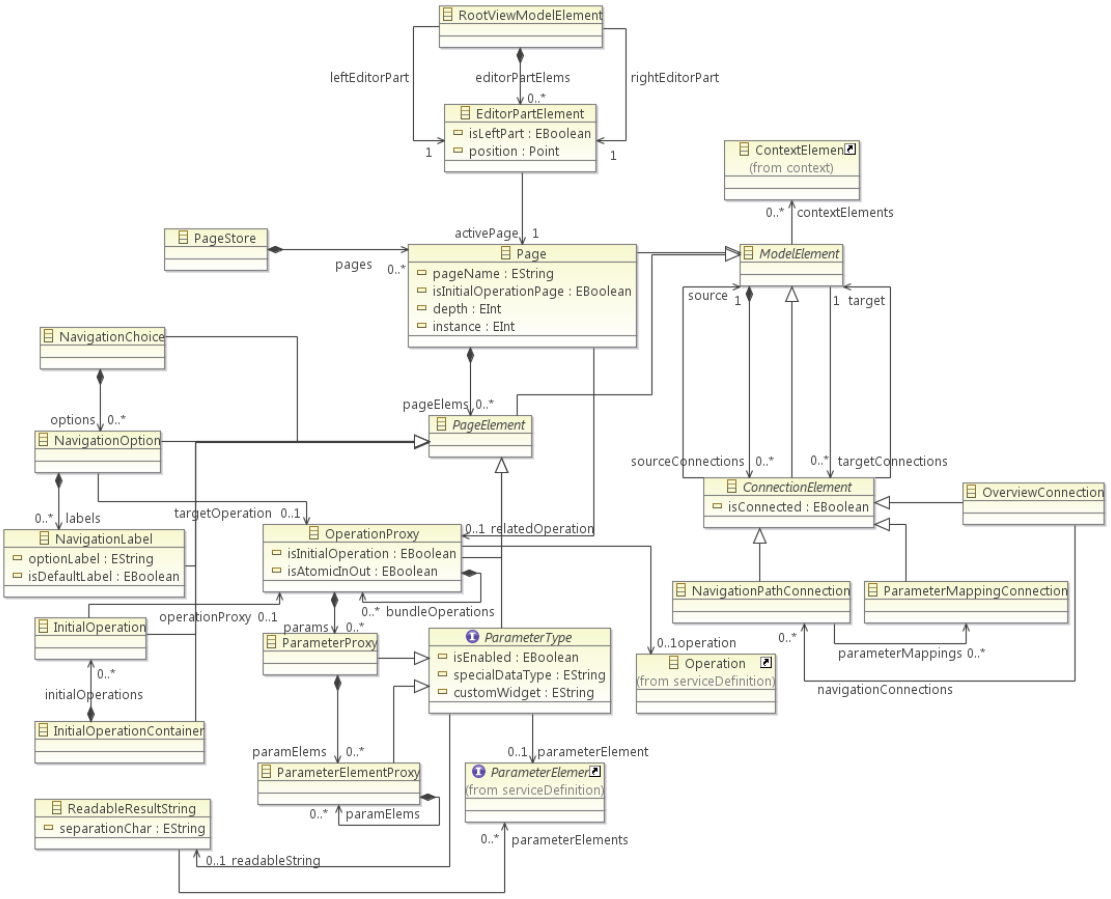


Abbildung 4.17.: Aufbau des Viewmodels

Der Aufbau des Viewmodels, welcher in Abbildung 4.17 dargestellt wird, bestimmt maßgeblich die Strukturen der Editoren. Demnach ist das *RootViewModelElement* als oberstes Modellelement zugleich das Eingabeelement für den *Navigation-Editor*, der in Abbildung 4.14 dargestellt wird. Der Editor gliedert sich demnach in einen linken und rechten Editorbereich (*EditorPartElement*), wobei jeder Bereich zur Darstellung einer angelegten Anwendungsseite (*Page*) dient. Eine Anwendungsseite ist wiederum ein Container für Elemente (*PageElement*), die zum Abbilden relevanter Daten zur Beschreibung der Dienstannotationen und zur Darstellung von Dienstelementen aus dem Servicemodel dienen. Über die Stellvertreterobjekte *OperationProxy*, *ParameterProxy* und *ParameterElementProxy* wird so ein direkter Bezug zum Servicemodel gehalten, ohne eine direkte Manipulation der entsprechenden Dienstelemente zuzulassen.

Der Overview-Editor nutzt das Modellelement *PageStore*, welches alle Anwendungsseiten enthält, als Eingabe, um eine Anwendungsübersicht mit allen Seiten und Transitionen der Anwendung darzustellen. Seitenelemente(*PageElement*), die vom Navigation-Editor für eine detaillierte Ansicht einer Anwendungsseite genutzt werden, werden in der Ansicht des Overview-Editors nicht berücksichtigt.

Da die graphischen Editoren zur Darstellung von Beziehungen zwischen Objekten auf Verbindungslinien zurückgreifen, wird diese Funktionalität im Viewmodel über die abstrakte Klasse *ConnectionElement* ermöglicht. Im Modell wird zwischen drei unterschiedlichen, graphischen Objektverbindungen unterschieden. Während die *NavigationPathConnection* und *ParameterMappingConnection* zur Beschreibung und Darstellung von Navigations- und Datenflüssen dienen, ist die *OverviewConnection* eng mit den Navigationspfaden verbunden und dient der Darstellung von Transitionen in der Ansicht des Overview-Editors.

4.2.4. Konzeption der Verarbeitungslogik

Die Verarbeitungslogik des Editors ist durch den dargestellten Informationsfluss in Abbildung 4.13 beschrieben. Der erste Schritt einer Anwendungserstellung besteht demnach im Importieren einer funktionalen Schnittstellenbeschreibung in Form einer WSDL-Datei. Zu diesem Zweck soll ein *WSDL-Parser* erstellt werden.

Die Aufgabe des *WSDL-Parsers* besteht darin, der Anwendung relevante Informationen funktionaler Dienstbeschreibungen aus WSDL-Dateien zur Verfügung zu stellen. Hierbei werden die komplex verschachtelteten Informationen aus der Dienstbeschreibung zunächst vereinfacht und daraufhin in das abstrakte Servicemodel übertragen. Abbildung 4.18 gibt einen Überblick über den Aufbau des WSDL-Parsers.

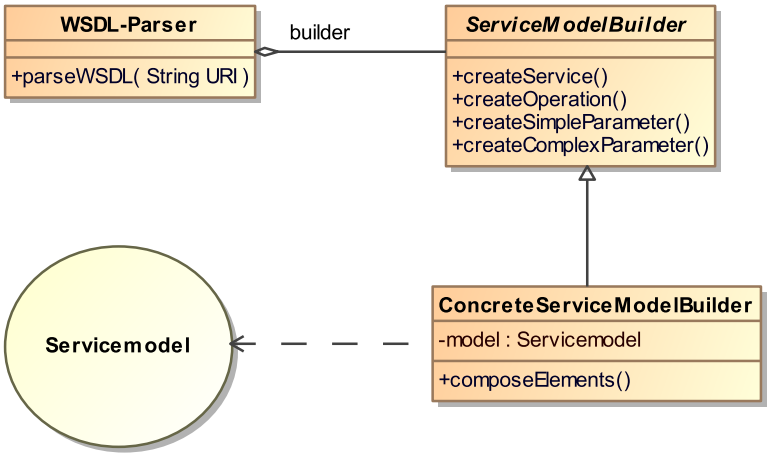


Abbildung 4.18.: Klassendiagramm des WSDL-Parsers

Da der WSDL-Parser eine Instanz des Servicemodels aus den einzelnen Dienstelementen der Dienstbeschreibung zusammensetzt und auch eine Unterstützung ähnlicher Dienstmodelle gewährleistet werden soll, wird für die Umsetzung des Parser das Erbauer-Entwurfsmuster

angewendet. Der Import einer WSDL-Dienstbeschreibung wird über die Methode *parseWSDL* unter Angabe einer URI angestoßen. Daraufhin werden die Informationen der Dienstbeschreibung zunächst in Objektdaten überführt und mit Hilfe des *ServiceModelBuilders* können die einzelnen Elemente des Servicemodels erzeugt und zusammengesetzt werden. Abgeschlossen wird der Importvorgang durch den Aufruf der Methode *composeElements*, welche die erzeugten Dienstelemente in das Servicemodel eingliedert und die Informationen der Anwendung somit global zur Verfügung stellt.

Nachdem dem Nutzer die Dienstelemente in der Serviceansicht bereitgestellt wurden, kann dieser für einzelne Dienstoperationen Anwendungsseiten in den Editoransichten anlegen, um Dienstannotationen modellieren zu können. Im Sequenzdiagramm, welches in Abbildung 4.19 dargestellt ist, wird exemplarisch anhand des Operationsobjektes aus dem Servicemodel die Beziehung zwischen Modell- und Editorobjekten verdeutlicht.

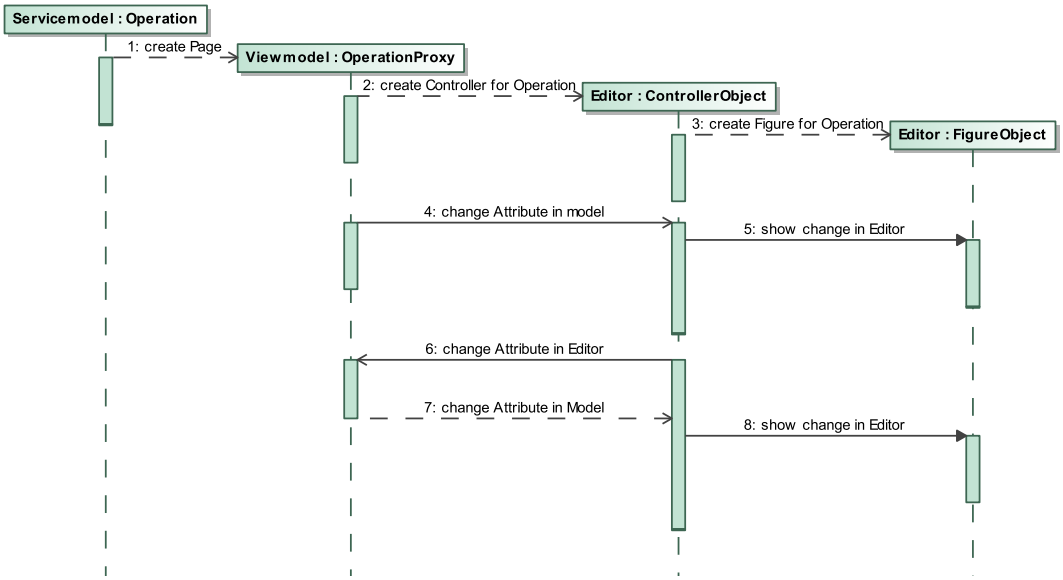


Abbildung 4.19.: Ablauf der Kommunikation zwischen Modell und Editor

Beim Anlegen einer neuen Seite für eine Operation des Servicemodels wird ein OperationProxy-Objekt im Viewmodel angelegt, um den Editoren später als Modellinformationen zur Verfügung zu stehen. Mit der Erzeugung eines neuen Objektes im Viewmodel wird automatische ein *Controllerobject* generiert, welches als Controller-Komponenten des MVC-Entwurfsmusters agiert und somit die Bindung zwischen dem Viewmodel und der Editordarstellung verwaltet. Hierzu erzeugt das *Controllerobject* ein *Figureobject* (View-Komponente), welches vom Editor zur Darstellung der Operation genutzt wird. Änderungen von Objektattributen im Viewmodel werden vom *Controllerobject* über einen Benachrichtigungsmechanismus abgefangen, ausgewertet und gegebenenfalls an das *Figureobject* weiterleiten, um die Änderung im Modell auch im Editor zu aktualisieren. Änderung von Objektattributen im Editor werden direkt über das *Controllerobject* kommuniziert und an das Viewmodel weitergegeben. Erst nachdem die Änderung im Viewmodel durchgeführt wurde, wird diese wieder mit Hilfe des Benachrichtigungsmechnismus über das *Controllerobject* an das *Figureobject* übertragen.

Nachdem eine Anwendung über die Editoransichten fertig modelliert wurde, kann diese mit Hilfe des *Annotation-Handlers* für die weiteren Prozessschritte zur Generierung interaktiver, dienstbasierter Anwendungen veröffentlicht werden. Der *Annotation-Handler* stellt der Anwendung die Funktionalitäten zum Exportieren und Importieren von Dienstannotationen zur Verfügung. Hierzu ist die Anwendung über einen Verzeichnisdienst mit einem Annotation-Repository verbunden, in welchem die mit dem Editor modellierten Anwendungen abgespeichert und aus dem diese Anwendungen wieder geladen werden können. Abbildung 4.20 gibt einen Überblick über den Aufbau des Annotation-Handlers.

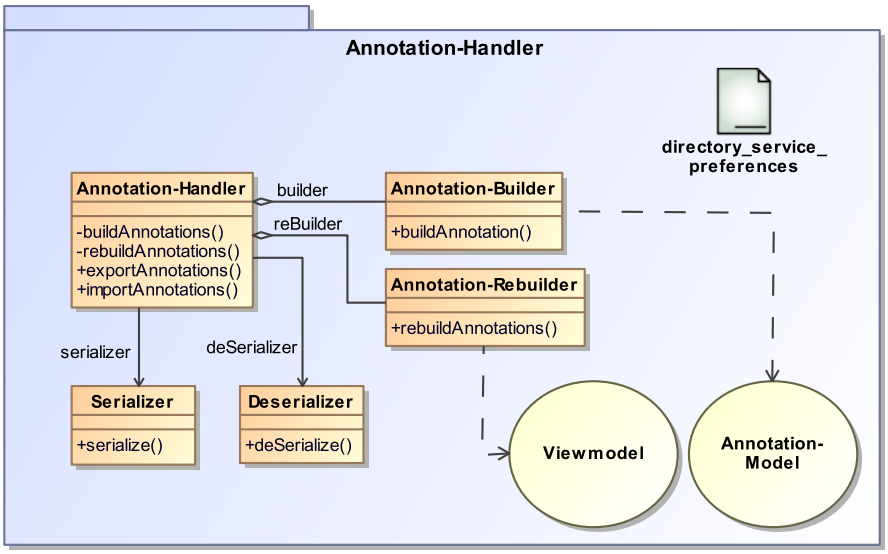


Abbildung 4.20.: Klassendiagramm der Annotation-Handler Komponente

Die Klasse *Annotation-Handler* bietet über die öffentlichen Methoden *exportAnnotations()* und *importAnnotations()* die Möglichkeit den Export beziehungsweise Import anzustoßen. Wird der Export der Dienstannotationen aufgerufen, so werden die dazu notwendigen Informationen aus dem Viewmodel, welches die modellierten Informationen der Anwendung enthält, abgeleitet und mit Hilfe des *Annotation-Builders* in Dienstannotationen des zugrundeliegenden *Annotation-Models* überführt. Die auf diese Weise erzeugte Instanz des *Annotation-Models* wird an einen Verzeichnisdienst übertragen und von diesem mit zusätzlichen Metainformationen im Annotation-Repository abgelegt. Die Informationen zum Verzeichnisdienst liegen der Anwendung in konfigurierbarer Form vor.

Bevor der Import von Dienstannotationen und damit verbunden das Laden einer Anwendung durchgeführt werden kann, wird über den Verzeichnisdienst zunächst die URI einer gewünschten Annotationsdatei angefordert. Mit der Dateiangabe wird dann mit Hilfe des *Deserializers* die Instanz des *Annotation-Models* rekonstruiert und aus den Dienstannotationen dieser Modelinstanz erzeugt der *Annotation-Rebuilder* Objekte des Viewmodels. Auf diese Weise wird die modellierte Anwendung im Viewmodel und somit auch in den Editoransichten wiederhergestellt.

4.2.5. Zusammenfassung

Basierend auf den aufgestellten Anforderungen wurde ein System entworfen, das eine graphische Modellierung von interaktiven, dienstbasierten Anwendungen ermöglicht. Mit dem Export von Anwendungsbeschreibungen in das in Kapitel 4.1 konzipierte erweiterte ServFace Annotation Model, wurde weiterhin eine Schnittstelle zur Integration des Editors in den Gesamtansatz von [Fel11] geschaffen.

5. Implementierung

Dieses Kapitel beschäftigt sich mit der Konzeptumsetzung und beschreibt die technische Realisierung eines graphischen Editors zur Beschreibung interaktiver, dienstbasierter Anwendungen mit Hilfe der in Abschnitt 4.1 konzipierten Dienstannotationen. Hierzu wurde für die Umsetzung das im vorherigen Kapitel vorgestellte System implementiert. Der bei dieser Umsetzung entstandene Editor geht über einen Prototypen hinaus, da der Funktionsumfang des Softwarewerkzeugs bereits sämtliche Schritte zum praktischen Erzeugen und Bereitstellen von Anwendungsbeschreibungen abdeckt. Im Folgenden wird auf die Implementierung einzelner Systemkomponenten eingegangen, wobei nur auf zentrale Aspekte eingegangen wird.

5.1. Umsetzung der Systemkomponenten

Aufgrund der speziellen Anforderungen und den konzipierten Bedienungsmechanismen kann für die Umsetzung des Editors auf keine bestehende und quelloffene Softwarekomponente zurückgegriffen werden. Bis auf den WSDL-Parser werden daher alle Bestandteile des Softwarewerkzeugs komplett eigenständig entwickelt. Für die Umsetzung sämtlicher Systemkomponenten wird die Programmiersprache Java und als Entwicklungsumgebung Eclipse 3.5 Galileo[8] verwendet. Um eine modulare Implementierung zu begünstigen und auf eine erprobte Anwendungsbasis zurückgreifen zu können, wird der Editor als Rich Client Platform[9](RCP) Anwendung realisiert. RCP stellt als Framework eine Infrastruktur zur Realisierung von modularen Desktop-Anwendungen auf OSGi-Basis[10] zur Verfügung, kann aber mit Hilfe des Frameworks Rich Ajax Platform[11] (RAP) auch als Webanwendung veröffentlicht werden. Vor allem aber die Bereitstellung ausgereifter Basiskomponenten für GUI-Anwendungen, wie beispielsweise Editoren und speziellen Ansichten, sind ausschlaggebend für die Entscheidung.

Für die Umsetzung der drei konzipierten Modelle *Servicemodel*, *Viewmodel* und *DeltaServFaceModel* wird das Eclipse Modeling Framework (EMF)[12] in der Version 3.5 verwendet. Neben der Bereitstellung verschiedener Werkzeuge zur graphischen Erstellung und Validierung von Modellen bietet EMF die Möglichkeit der Quellcodegenerierung. Dadurch können Modelländerungen mit Hilfe des graphischen Editors vorgenommen werden ohne

[8] http://www.eclipse.org/galileo/
[9] www.eclipse.org/rcp/
[10] http://www.osgi.org/
[11] http://www.eclipse.org/rap/
[12] www.eclipse.org/emf/

den Modellquellcode manuell anpassen zu müssen. Ein weiterer Vorteil von EMF ist die automatische Generierung eines Benachrichtigungsmechanismus, welcher die Reaktion auf Änderungen innerhalb eines Modells deutlich vereinfacht. Mit dem EMF Persistence Framework stellt EMF darüber hinaus eine Möglichkeit zur Serialisierung und Deserialisierung der Modelle im XML- oder XMI-Format.

Neben dem Editor und den Modellen wird auch die benötigte Geschäftslogik des in Absatz 3.1 beschriebenen Anwendungsszenarios erstellt. Hierzu soll das Eclipse-Plugin Web Tools Platform[13] mit Apache Axis 2[14] in der Version 1.4.1 zur werkzeuggestützten Erstellung und Veröffentlichung der beiden im Szenario beschriebenen Dienste *PatientService* und *MedicamentService* verwendet werden. Die Dienste werden nach außen durch funktionale Schnittstellenbeschreibungen der WSDL-Spezifikation [Chr01] in der Version 1.1 repräsentiert und stehen zum Testen während der Implementierung des Editors zur Verfügung. In Absatz 6.1 wird der umgesetzte *PatientService* darüber hinaus exemplarisch zur Evaluierung des Editors herangezogen.

5.1.1. WSDL-Parser

Um dem Editor die Informationen aus den funktionalen Schnittstellenbeschreibungen zur Verfügung zu stellen, wird der WSDL-Parser, welcher im Rahmen der Belegarbeit [Mar09] und [Ber09] entwickelt wurde, aufgegriffen und erweitert. Der WSDL-Parser basiert auf der Klassenbibliothek Web Services Description Language for Java Toolkit[15] (WSDL4J). WSDL4J bietet einen reichhaltigen Funktionsumfang zum erzeugen, repräsentieren und manipulieren von WSDL-Dateien. Für die Umsetzung wurde der integrierte WSDL-Parser der Klassenbibliothek, der durch die Klasse *WSDL4JParser* repräsentiert wird, verwendet. Listing 5.1 zeigt die Anwendung des Parsers.

```
1 ServiceDefinitionBuilder sdBuilder = new ServiceDefinitionBuilder();
2 WSDL4JParser wsdlParser;
3 try {
4   wsdlParser = new WSDL4JParser(sdBuilder);
5   wsdlParser.parseWSDL("http://localhost:8080/PatientService/services/Patient_Service?wsdl");
6   ServiceModel sModel = sdBuilder.getServiceModel();
7 } catch (WSDLException e) {
8   e.printStackTrace();
9 }
```

Listing 5.1: Anwendung des WSDL-Parsers

Um die Informationen einer WSDL-Datei als Instanz eines individuellen Dienstmodells zu erhalten, wird dem Parser ein *ServiceDefinitionBuilder* übergeben. Dieses konkrete Builder-Objekt (vgl. [GHJV95]) erzeugt die einzelnen Dienstobjekte beim Parsen aus den Informationen der WSDL-Datei und setzt diese zu einem Dienstmodell zusammen. Als Dienstmodell wird für die Umsetzung das *ServiceModel*, welches Teil des *AnnotatedServiceModel* ist und im Rahmen der Dissertation von [Fel11] entstanden ist, verwendet.

[13] http://www.eclipse.org/webtools/
[14] http://ws.apache.org/axis2/
[15] http://sourceforge.net/projects/wsdl4j/

5.1.2. Navigation-Editor und Overview-Editor

Zur Umsetzung der Editoren wird das Graphical Editing Framework[16] (GEF) in der Version 3.5 verwendet. GEF ist ein Eclipse-Projekt, das sich mit der Erstellung von graphischen Editoren auf Basis des Standard Widget Toolkits[17] (SWT) und der Eclipse-Workbench-API beschäftigt und ist damit ideal für die Erstellung von Editoren geeignet, die in Eclipse-RCP Anwendungen integriert werden können. Die Erstellung eines Editors mit GEF ist an die Struktur des Model-View-Controller-Paradigmas gebunden, um eine strikte Trennung zwischen dem zugrundeliegenden Datenmodell und der für die Darstellung genutzten Objekte zu schaffen. Als Datenmodell dient beiden Editoren das in Abschnitt 4.2.3 konzipierte *Viewmodel*. Um eine vollständige Darstellung des Viewmodels in den Editoransichten jederzeit zu ermöglichen, überwachen die Editoren das Modell auf die Erzeugung neuer Viewmodel-Objekte und erstellen mit Hilfe einer *EditPartFactory* entsprechende EditPart-Objekte (Listing 5.2).

```
1  public EditPart createEditPart(EditPart context, Object model){
2      EditPart part = null;
3
4      if(model instanceof RootViewModelElementImpl)
5        part = new RootViewModelElementEditPart();
6      else if(model instanceof PageImpl)
7        part = new PageContentEditPart();
8      ...
9      else if(model instanceof OperationProxyImpl)
10       part = new OperationEditPart();
11     else if(model instanceof ParameterProxyImpl)
12       part = new ParameterEditPart();
13
14     part.setModel(model);
15
16     return part;
17 }
```

Listing 5.2: Methode der EditPartFactory zur Erzeugung der EditParts

EditPart-Objekte entsprechen den Controllern des MVC-Paradigmas und werden als Mediator zwischen Modell- und Repräsentationsobjekten gesehen. Ein EditPart-Objekt ist für die Erstellung(*createFigure()*) und Aktualisierung(*refreshVisuals()*) eines Repräsentationsobjektes(*figure*) bei einer Änderung(*notifyChanged()*) der Eigenschaften des Modellobjektes zuständig. Listing 5.3 zeigt die Grundfunktionalitäten zur Verwaltung des Repräsentationsobjektes am Beispiel der Klasse *NavigationLabelEditPart*.

Für die Repräsentation von Objekten wurde im Rahmen des GEF-Projektes die Grafikbibliothek Draw2D entwickelt, welche einfache graphische Figuren wie beispielsweise Rechteck, Ellipse oder Label zur Verfügung stellt. Im Beispiel wird zur Darstellung eines *NavigationLabels* ein Label verwendet. Bei einer Aktualisierung, welche durch Änderungen im Modellobjekt aufgerufen wird, wird der Text des Labels durch den Wert des entsprechenden *NavigationLabels* aus dem Viewmodell-Objekt ersetzt.

[16] http://www.eclipse.org/gef/
[17] http://www.eclipse.org/swt/

```
1  public class NavigationLabelEditPart extends AbstractGraphicalEditPart implements Adapter{
2
3    private draw2d.Label figure;
4
5    protected IFigure createFigure() {
6      figure = new Label();
7      return figure;
8    }
9
10   protected void refreshVisuals() {
11     figure.getInputLabel().setText(((NavigationLabel)getModel()).getOptionLabel());
12   }
13
14   public boolean isAdapterForType(Object type) {
15     return type.equals( getModel().getClass());
16   }
17
18   public void notifyChanged(Notification n) {
19     refreshVisuals();
20   }
21   ...
22 }
```

Listing 5.3: NavigationLabelEditPart

Um den von EMF generierten Benachrichtigungsmechanismus zum Abfangen von Änderungen im Viewmodel nutzen zu können, implementiert die Klasse *NavigationLabelEditPart* hierzu die EMF Schnittstelle *Adapter*.

Zum Bereitstellen diverser Interaktionsmöglichkeiten für die Anwendungsmodellierung werden die im Konzept eingeführten Werkzeuge umgesetzt. Auch hierfür stellt das Graphical Editing Framework Basiskomponenten bereit, die eine Umsetzung vereinfachen.

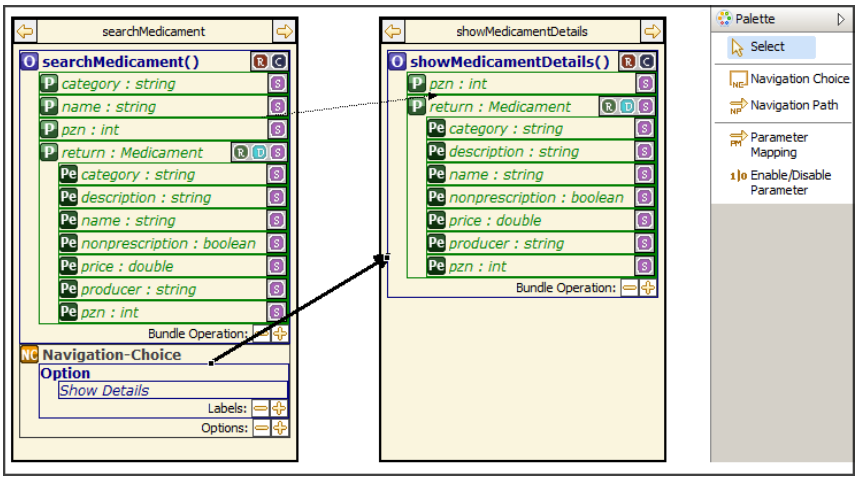


Abbildung 5.1.: Editoransicht des Navigation-Editors

Hierzu zählt beispielsweise eine Werkzeugpalette, die als Container zur graphischen Darstellung der Werkzeuge mit der Editoransicht verbunden ist. Über die Palette kann der Nutzer die zur Verfügung stehenden Werkzeuge aktivieren und jederzeit prüfen, welches

Werkzeug aktiv ist. Abbildung 5.1 zeigt die Editoransicht des Navigation-Editors mit Werkzeugpalette.

Zur funktionalen Umsetzung von Werkzeugen bietet GEF eine Vielzahl von abstrakten Werkzeugklassen mit Basisfunktionalitäten für verschiedene Werkzeugtypen, wobei zwischen Selektionswerkzeugen, objekterzeugenden Werkzeugen und Verknüpfungswerkzeugen unterschieden wird. Das Prinzip von Werkzeugen besteht darin, Nutzereingaben, die beispielsweise mit der Maus oder Tastatur durchgeführt werden, abzufangen und in Abhängigkeit des aktiven Werkzeugs in komplexere, gekapselte Operationen zu überführen. Die gekapselten Operationen werden als Request-Objekte zur Kommunikation mit den EditParts verwenden. Um ein EditPart für die Kommunikation mit Request-Objekten zu registrieren, müssen EditPolicies zur Festlegung des Editierverhalten definiert werden. Listing 5.4 demonstriert das Anlegen von EditPolicies am Beispiel der Klasse ParameterEditPart.

```
1 public class ParameterEditPart extends AbstractGraphicalEditPart implements NodeEditPart{
2   ...
3   protected void createEditPolicies() {
4     installEditPolicy(EditPolicy.GRAPHICAL_NODE_ROLE, new ParameterMappingEditPolicy());
5     installEditPolicy(EditPolicy.DIRECT_EDIT_ROLE, new DisableParameterEditPolicy());
6   }
7   ...
8 }
```

Listing 5.4: EditPolicies von ParameterEditPart

Demnach werden für ein ParameterEditPart zwei unterschieldiche Editierverhalten festgelegt. Um eine Anwendung des ParameterMapping-Werkzeugs zu ermöglichen, wird eine neue EditPolicy zur Behandlung von Request-Objekten, die vom ParameterMapping-Werkzeug initiiert werden, erstellt. Da das ParameterEditPart beim Erstellen von Parameterabbildungen als Knotenpunkt fungieren soll, wird dies durch die zusätzliche Angabe der Objektrolle festgelegt. Analog hierzu wird mit Hilfe der DisableParameterEditPolicy die Anwendung des DisableParameter-Werkzeugs, welches auf einem Auswahlwerkzeuges basiert, realisiert. Listing 5.5 zeigt die Umsetzung des Werkzeugs.

```
1 public class DisableParameterEditPolicy extends SelectionEditPolicy{
2
3   @Override
4   protected void showSelection() {
5     ParameterProxy pp = (ParameterProxy)getHost().getModel();
6     pp.setIsEnabled(!pp.isIsEnabled());
7   }
8 }
```

Listing 5.5: DisableParameterEditPolicy

Wenn das DisableParameter-Werkzeug aktiv ist und ein Parameter in der Editoransicht selektiert wird, führt der damit verbundene Methodenaufruf von *showSelection* dazu, dass die Eigenschaft *isEnabled* eines ParameterProxy-Objektes im Viewmodel negiert wird. So kann das Auswahlwerkzeug auf einfache Weise zum Manipulieren der Eigenschaft genutzt werden.

5.1.3. Annotation-Handler

Der Annotation-Handler umfasst alle Klassen, die mit der Serialisierung und Deserialisierung von modellierten Anwendungen zu tun haben. Abbildung 5.2 gibt einen Überblick über die Klassen und Methoden.

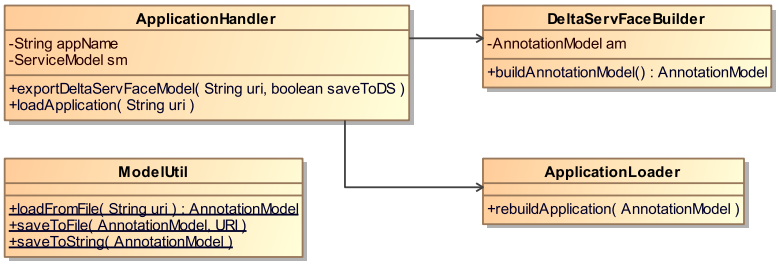


Abbildung 5.2.: Klassen zum Laden und Serialisieren von Anwendungen

- **ApplicationHandler**

  Die Klasse *ApplicationHandler* stellt dem Editor als Singleton diverse Methoden zur Verwaltung grundlegender Anwendungsinformationen zur Verfügung. Eine Aufgabe des Verwalterobjektes besteht demnach auch darin, das Speicher beziehungsweise Laden von Anwendungen zu steuern. Um eine modellierte Anwendung zu exportieren, wird die Methode *exportDeltaServFaceModel* über einen entsprechenden Exportdialog aufgerufen. Da ein lokales Speichern und das Persistieren in einem Repository unterstützt werden und beide Varianten eine unterschiedliche Routine verfolgen, gibt der Aufrufparamter *saveToDS* an, um welche Art der Persistierung es sich handelt. In beiden Fällen wird unter Verwendung des DeltaServFaceBuilder-Objektes das Annotationsmodell anhand der modellierten Dienstannotationen für die Anwendung abgeleitet. Wird die erstellte Anwendung lokal abgespeichert, so wird die Modellinstanz mit Hilfe der statischen Methode *saveToFile* der Klasse ModelUtil auf dem lokalen System erstellt. Das Abspeichern im Repository wird über einen Aufruf des Verzeichnisdienstes, welcher im Verlauf noch vorgestellt wird, realisiert. Hierzu wird die Modellinstanz mit der Methode *saveToString* als String-Wert repräsentiert und in dieser Form dem Dienst zur weiteren Verarbeitung übergeben.

  Das Laden einer Anwendung wird mit der Methode *loadApplication* umgesetzt. Unter Angabe der URI einer zu ladenden Datei wird mit Hilfe der ModelUtil-Methode *loadFromFile* das Annotationsmodell wiederhergestellt. Das ApplicationLoader-Objekt kann daraufhin aus den Informationen des Annotationsmodells die Anwendung im Editor vollständig rekonstruieren.

- **DeltaServFaceBuilder**

  Die Klasse *DeltaServFaceBuilder* erstellt über den Methodenaufruf von *buildAnnotationModel* zu einem beliebigen Zeitpunkt der Anwendungsmodellierung die Anwendungsbeschreibung. Hierzu wird eine Instanz des erweiterten ServFace Annotation-

Models erzeugt, indem aus den modellierten Informationen, die im Viewmodell abgebildet sind, Dienstannotationen abgeleitet werden. Zusätzlich zu den Dienstannotationen, die im Viewmodell abgebildet werden und bereits eine vollständige Anwendungsbeschreibung darstellen, werden ein Teil der ursprünglichen ServFace-Annotationen unterstützt. Diese werden separat vom Viewmodell gehalten.

- **ApplicationLoader**

  Die Klasse *ApplicationLoader* stellt das Gegenstück zum DeltaServFaceBuilder dar. Anhand einer AnnotationModel-Instanz wird über die Methode *rebuildApplication* das Viewmodell aus den Informationen der Dienstannotationen rekonstruiert.

- **ModelUtil**

  Die Klasse *ModelUtil* kapselt die Funktionalitäten zum Abspeichern und Laden von Dateien. Hierzu werden die vom Eclipse Modelling Framework zur Verfügung gestellten Werkzeuge zum Serialisieren und Deserialisieren eingesetzt.

Da zur Unterstützung der Prozesskette des übergreifenden Ansatzes, welche in Absatz 1.2 vorgestellt wurde, eine Veröffentlichung von modellierten Anwendungen in einem speziellen Repository für Dienstannotationen vorgesehen ist, soll neben dem lokalen Dateiexport auch die zentrale Veröffentlichung mit Hilfe eines Verzeichnisdienstes realisiert werden. Der Dienst soll sowohl eine Instanz des Annotationsmodells entgegennehmen, um diese im Repository abzuspeichern, als auch das Suchen nach abgespeicherten Anwendungen ermöglichen. Hierzu stellt der Dienst die in Abbildung 5.3 dargestellten Methoden zur Verfügung.

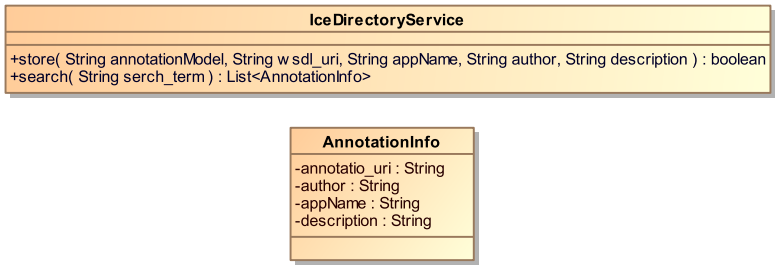


Abbildung 5.3.: Directory Service

Zum Veröffentlichen einer Anwendung dient die Dienstoperation *store*. Um die Operation aufrufen zu können, muss die Anwendung mit Hilfe des DeltaServFaceBuilders bereits als Annotationsmodell vorliegen, da die Modellinstanz als String-Wert übergeben wird. Um eine Zuordnung der Anwendungsbeschreibung zur zugehörigen WSDL-Datei zu gewährleisten, wird die URI der funktionalen Dienstbeschreibung, welche über den Application-Handler direkt verfügbar ist, als zweiter Aufrufparameter übergeben. Auch der Anwendungsname wird vom ApplicationHandler verwaltet und übergeben. Die Aufrufparameter *author* und *description* sind optional und dienen als zusätzliche Metainformationen einer

detaillierteren Beschreibung, um beispielsweise unterschiedliche Versionen einer Anwendung unterscheiden zu können. Ein Aufruf der Dienstoperation bewirkt, dass das Annotationsmodell als XMI-Datei in das Repository-Verzeichnis abgespeichert wird. Hierzu wird als eindeutiger Bezeichner für den Dateinamen ein Zeitstempel verwendet. Um später über den Verzeichnisdienst eine Suche nach abgespeicherten Dateien zu ermöglichen, werden die zusätzlich übergebenen Metainformationen *wsdl_uri*, *appName*, *author* und *description* zusammen mit der URI der angelegten Datei in einer Datenbank abgelegt.

Um eine Anwendung zu laden, stellt der Verzeichnisdienst die Operation *search* zur Verfügung. Die Operation sucht anhand des übergebenen Suchbegriffs in der Datenbank nach passenden Einträgen. Hierbei werden für die Suche alle Datenbankfelder zur Beschreibung der Anwendungen berücksichtigt. Die Rückgabe der Suchergebnisliste erfolgt in Form einer Liste von Objekten des Typs *AnnotationInfo*. Die Liste enthält sämtliche Informationen, um den weiteren Importprozess clientseitig umzusetzen, da die AnnotationInfo-Objekte die URI der zugehörigen Instanz des Annotationsmodells beinhalten. Mit Hilfe der Methode *loadApplication* des ApplicationHandlers kann so das Laden einer Anwendung angestoßen werden.

6. Evaluierung

Nachdem im Kapitel 5 die Umsetzung des Editors zur Beschreibung von interaktiven, dienstbasierten Anwendungen mittels Dienstannotationen vorgestellt wurde, soll der entwickelte Prototyp in diesem Kapitel zunächst hinsichtlich der Funktionalität am Anwendungsszenario validiert und bezüglich der Benutzbarkeit mit Hilfe eines Usability-Tests detailliert analysiert werden. Im Anschluss daran soll der gesamte Ablaufprozess des Gesamtkonzeptes von der Anwendungsmodellierung bis zur Ausführung der Anwendung auf dem Zielgerät validiert werden. Abschließend werden die in Abschnitt 3.2 aufgestellten Anforderungen bezüglich ihrer Erfüllung untersucht.

6.1. Validierung des Editors am Anwendungsszenario

Das Anwendungsszenario, welches in Kapitel 3.1 vorgestellt wurde, beschreibt eine Anwendung zur mobilen Verwaltung von Patientendaten und Medikamentenbestellungen. Zur Bereitstellung der Basisfunktionalitäten der Anwendung wurden hierzu je ein Dienst für die Patientenverwaltung und Medikamentenbestellung erstellt, welche über ihre WSDL-Beschreibungen nach außen repräsentiert werden.

Im Folgenden soll veranschaulicht werden, wie ein Benutzer durch Verwendung des Editors den im Anwendungsszenario beschriebenen Teil der Anwendung zur Patientenverwaltung umsetzen kann. Nach dem Start des Editors wird zunächst eine neue Anwendung angelegt(Abbildung 6.1). Hierbei findet die WSDL-Datei des Patientendienstes seine erste Verwendung, da eine Anwendung genau einem initialen Dienst zugeordnet ist. Zu einem späteren Zeitpunkt können der Anwendung aber jederzeit weitere Dienste hinzugefügt werden.

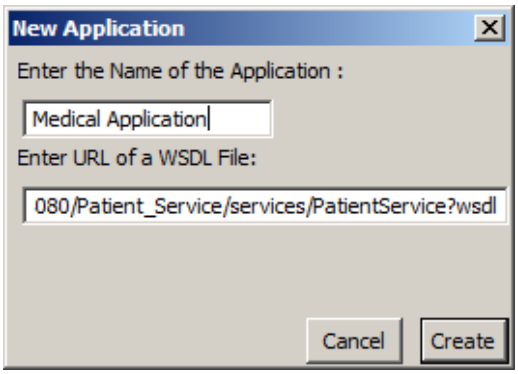


Abbildung 6.1.: Anlegen einer neuen Anwendung

Nachdem die neue Anwendung angelegt wurde, wird in der Dienstübersicht durch eine Baumansicht die Struktur des Patientendienstes mit den verfügbaren Dienstoperationen dargestellt. Über das Kontextmenü können in einem nächsten Schritt Anwendungsseiten für die Dienstoperationen *addPatient* und *searchPatient* angelegt werden. Hierzu bietet sich die Arbeit im Overview-Editor an, da dort alle angelegten Anwendungsseiten übersichtlich dargestellt und als initiale Anwendungszustände ausgezeichnet werden können. Abbildung 6.2 zeigt den Fortschritt der Anwendungsmodellierung im Overview-Editor.

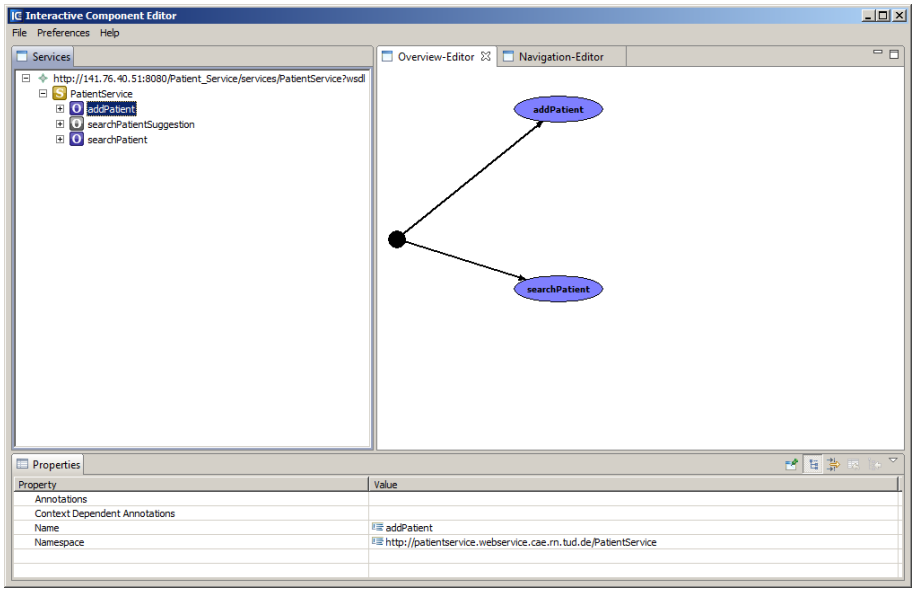


Abbildung 6.2.: Anlegen initialer Anwendungsseiten im Overview-Editor

Die weiteren Schritte lassen sich nur im Navigation-Editor umsetzen. Da das Sucheingabefeld für den Aufruf der Operation *searchPatient* und die zurückgegebene Ergebnisliste atomar auf einer Seite dargestellt werden sollen, kann hierzu die Eigenschaft *IsAtomicInOut* in der Eigenschaftenansicht für die Dienstoperation gesetzt werden. Um die Ergebnisliste mit den umfangreichen Patientendaten vereinfacht und für den Nutzer besser lesbar darzustellen, kann auf den komplexen Rückgabewert der Operation die Funktion *ReadableString* angewendet werden, indem festgelegt wird, dass die Patientendaten in der Liste nur durch den Vor- und Nachnamen repräsentiert werden(Abbildung 6.3).

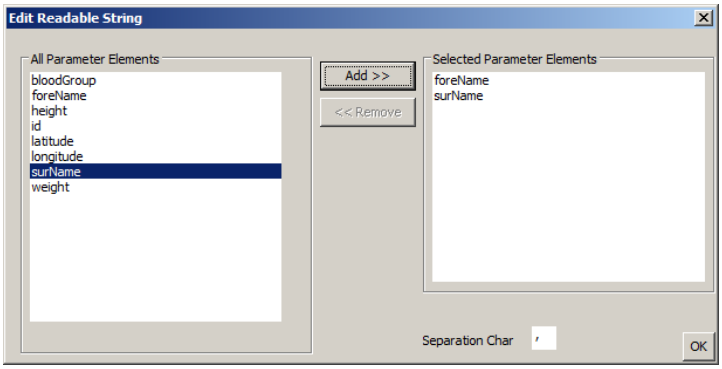


Abbildung 6.3.: Anpassung der Darstellung eines komplexen Datentypen

Das Szenario sieht weiterhin zwei unterschiedliche Navigationsflüsse von der Anwendungsseite *searchPatient* zu *addPatient* vor, um einen neuen Patienten anzulegen beziehungsweise die Daten eines bestehenden Patienten editieren zu können. Hierzu bietet der Navigation-Editor geeignete Werkzeuge. Zunächst wird auf der Anwendungsseite *searchPatient* mit dem NavigationChoice-Werkzeug ein NavigationChoice-Objekt erstellt, welches es ermöglicht die zwei gewünschten Navigationsoptionen mit Bezeichnern für die zu generierenden Navigationselemente zu erstellen. Daraufhin kann das NavigationConnection-Werkzeug genutzt werden, um von den Navigationsoptionen je eine Verknüpfung zur Anwendungsseite *addPatient* zu erstellen. Um über die Operation *addPatient* ein Editieren von Patientendaten zu ermöglichen, müssen für den Navigationsübergang zum Editieren zusätzlich Parameterabbildungen definiert werden, die das Weitergeben der Daten des selektierten Patienten aus der Ergebnisliste gewährleisten. Zu diesem Zweck wird das ParameterMapping-Werkzeug genutzt, mit dem sich Verknüpfungen zwischen den Ausgabeparametern der Operation *searchPatient* und den Eingabeparametern der Operation *addPatient* definieren lassen. Abbildung 6.4 zeigt die modellierten Navigations- und Datenflüsse in der Ansicht des Navigation-Editors.

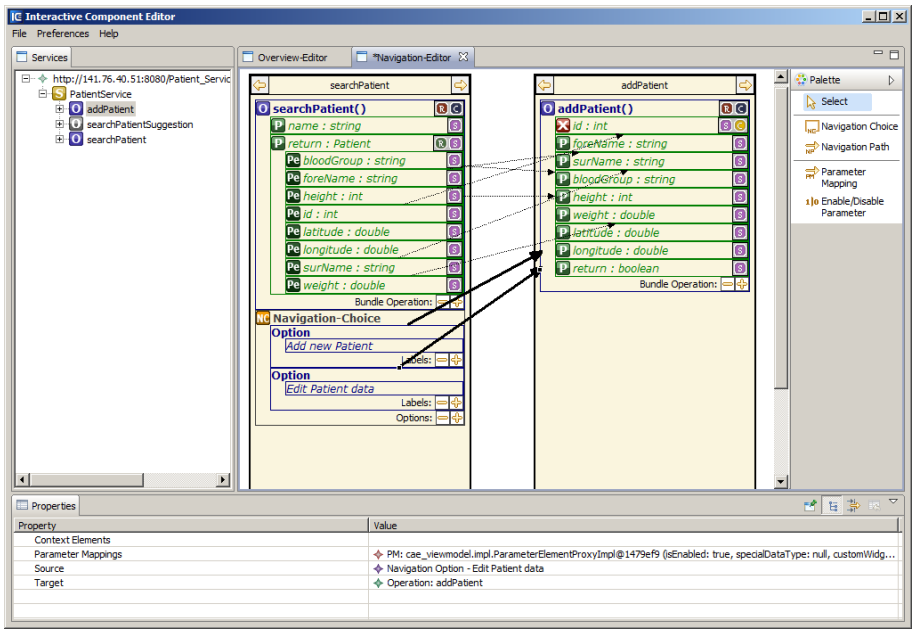


Abbildung 6.4.: Modellierte Navigations- und Datenflüsse im Navigation-Editor

Da das Indexierungsverfahren der Einträge von Patientendaten automatisch von einer Datenbank verwaltet wird, soll für den Parameter *id* der Operation *addPatient* kein Eingabefeld generiert werden. Dies wird einfach durch Selektierung des Parameters mit dem DisableParameter-Werkzeugs erreicht. Auch für die Parameter *latitude* und *longitude* soll der Nutzer keine manuellen Eingabemöglichkeiten haben. Stattdessen werden die GPS-Werte beim Dienstaufruf vom mobilen Endgerät zur Verfügung gestellt. Über die Funktion *Special Datatype* kann diese Eigenschaft festgelegt werden, wie in Abbildung 6.5 dargestellt.

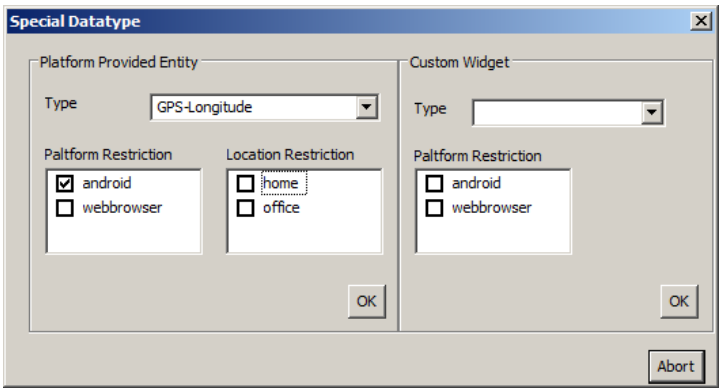


Abbildung 6.5.: Festlegung eines speziellen Datentypen

Damit sind alle Anforderungen des Anwendungsszenarios abgedeckt. Abbildung 6.6 zeigt die dargestellte Anwendungsstruktur der modellierten Anwendung. Neben den beiden Startnavigationen, die den Aufruf der Anwendungsseiten *addPatient* und *searchPatient* vom Startzustand der Anwendung darstellen, wird eine Transition zwischen den beiden Anwendungsseiten dargestellt, welche die erstellten Navigationen abbildet. Die zusätzliche Anwendung der Parameterabbildungen ist durch ein entsprechendes Symbol gekennzeichnet.

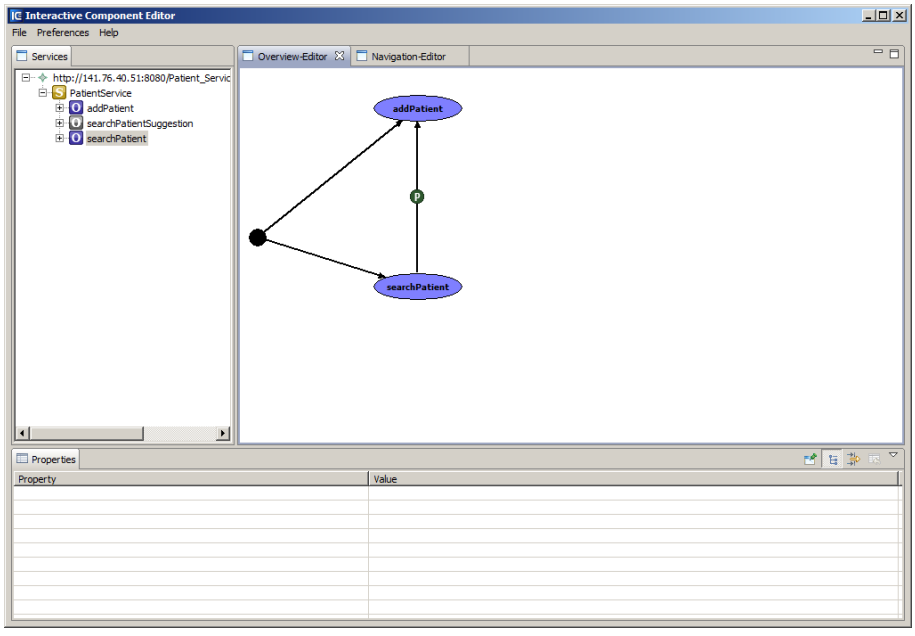


Abbildung 6.6.: Anwendungsstruktur im Overview-Editor

Die modellierte Anwendung zur mobilen Patientenverwaltung kann nun im Editor exportiert werden. Hierbei werden alle Informationen der Anwendungsbeschreibung in Form der in Kapitel 4.1 konzipierten Dienstannotationen serialisiert. Listing A.2 zeigt die erzeugte Instanz der in diesem Absatz entworfenen Anwendung zur Patientenverwaltung.

6.2. Usability-Test

Der Usability-Test soll dabei helfen, eine eindeutige Aussage über die Benutzbarkeit beziehungsweise Benutzerfreundlichkeit des Editors treffen zu können und dabei gelungene oder weniger gelungene Bedienungsmechanismen herauszustellen. Hierzu mussten die Testteilnehmer, nach einer kurzen Einführung, anhand einer Aufgabenbeschreibung mit dem Editor eine dienstbasierte Anwendung modellieren und im Anschluss die Benutzerfreundlichkeit durch die Beantwortung eines Fragebogens einschätzen. Der Test wurde mit insgesamt 16 Probanden durchgeführt, wobei alle Testteilnehmer über Fachkenntnisse aus dem IT-Umfeld verfügten. Aufgrund der hohen Anzahl an Testteilnehmern konnte der Test in zwei für eine Testauswertung repräsentative Gruppen mit je 8 Probanden aufgeteilt werden. Obwohl beide Gruppen das gleiche Testszenario modellieren sollten, wurden die Aufgabenstellung unterschiedlich formuliert. Während Gruppe 1 eine detaillierte Schritt-für-Schritt Anleitung in Textform erhalten hat, bekam Gruppe 2 eine ergebnisorientierte Szenarienbeschreibung mit einer Abbildung der zu realisierenden Anwendungsstruktur. Hierbei war das Ziel herauszufinden, ob sich der Editor für unerfahrene Nutzer intuitiv und zielgerichtet bedienen lässt. Der Fragebogen war für beide Gruppen identisch. Die Aufgabenstellungen der beiden Gruppen und der Fragebogen sind im Anhang A.1 zu finden. Im Folgenden werden die Fragebögen der Probanden ausgewertet.

Die erste Frage, die betrachtet werden soll, bezieht sich auf den Aufbau beziehungsweise die Gesamtstruktur des Editors. Das Diagramm in Abbildung 6.7 zeigt, dass der Aufbau des Editors von allen Probanden positiv bewertet wurde, wobei die Bewertungen von Gruppe 1 eine bessere Einschätzung darlegen.

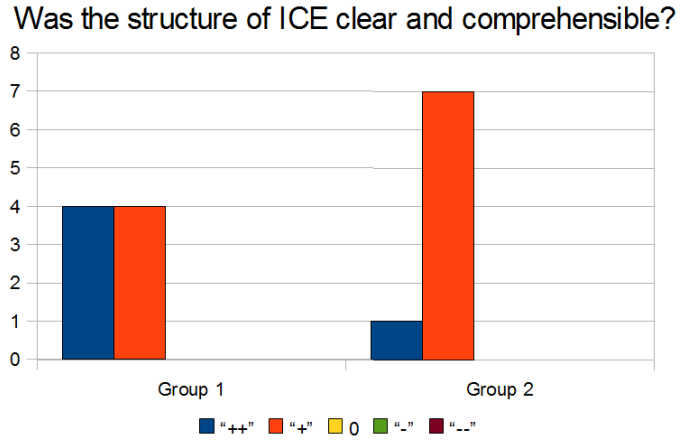


Abbildung 6.7.: Bewertung des Aufbaus und der Struktur des Editors

Dies ist vermutlich auf die detailliertere Aufgabenbeschreibung von Gruppe 1 zurückzuführen, da diese ein geführtes und weniger selbständiges Arbeiten mit dem Editor bewirkt. Unterstützt wird diese Vermutung durch die Auswertung der nachfolgenden Fragen. Während 7 Probanden aus Gruppe 1 angaben alle Editorfunktionen zur Erfüllung der Aufgabe gefunden zu haben, waren es in der Gruppe 2 nur 5 Probanden. Die übrigen Testteilneh-

mer hatten in der Regel Probleme mit dem Anwenden des ParameterMapping-Werkzeugs, da dieses erst sichtbar wird, nachdem ein zugehöriger Navigationspfad selektiert wurde. Auch bei der Frage, ob funktionale Probleme bei der Arbeit mit dem Editor aufgetreten sind, äußerten sich die Gruppen vermutlich aufgrund der unterschiedlichen Formulierung der Aufgabe differenziert. Aus Gruppe 1 konnte kein Proband auf funktionale Probleme verweisen, während 4 Probanden aus Gruppe 2 Probleme mit verschiedenen Programmfunktionen hatten. Kritikpunkte waren hier, dass das Programm über keine Undo-Funktion verfügt und das Löschen von angelegten Anwendungsseiten in der für den Usability-Test vorliegenden Version des Editors noch nicht möglich war.

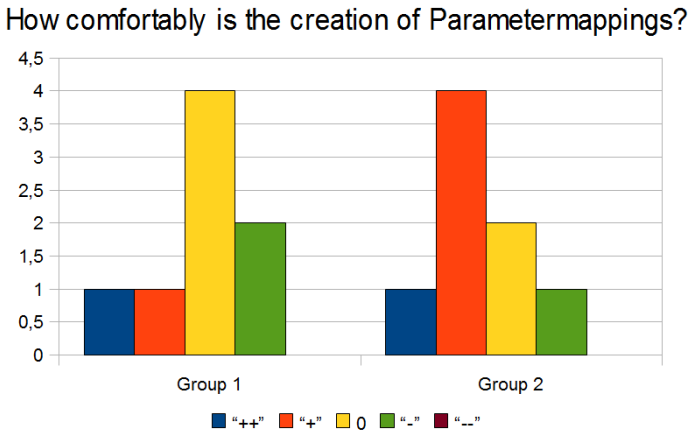


Abbildung 6.8.: Bewertung der Benutzerfreundlichkeit des ParameterMapping-Werkzeugs

Der bereits herausgestellte Kritikpunkt mit dem ParameterMapping-Werkzeug wird auch durch die nachfolgende Frage aufgegriffen. Das Diagramm 6.8 zeigt deutlich, dass die Einschätzung der Benutzerfreundlichkeit des ParameterMapping-Werkzeugs eine deutliche Streuung aufweist.

Abschließend wurden die Testteilnehmer noch zu einer Einschätzung von relevanten Programmeigenschaften des Editors gebeten. Die hierzu in Diagramm 6.9 dargestellte Auswertung zeigt, dass die Programmeigenschaften allgemein positiv beurteilt wurden. Vor allem die mögliche Effizienz bei der Modellierung von Anwendungen konnte die Probanden überzeugen. Dies zeigt deutlich, dass der Editor als Werkzeug zur vereinfachten, zielgerichteten Modellierung von dienstbasierten Anwendungen wahrgenommen wurde.

Zusammenfassend kann aus der Auswertung des Usability-Tests geschlossen werden, dass der umgesetzte Editor die Anforderung der Benutzbarkeit erfüllt. Alle Testteilnehmer waren nach einer kurzen Einführung mit dem Editor in der Lage die gestellte Aufgabe umzusetzen. Hierbei muss insbesondere herausgestellt werden, dass die Probanden der Gruppe 2 bezüglich der korrekten Anwendungsmodellierung anhand der Aufgabenstellung ebenso gute Ergebnisse erzielt haben, wie die Teilnehmer der Gruppe 1. Aufgrund der freien, zielorientierten Aufgabenformulierung für die Gruppe 2, wird durch diese Tatsache deutlich, dass der Editor ein selbstständiges und zielgerichtetes Arbeiten ermöglicht. Ein weiterer Beleg dafür, dass der Editor allgemein positiv aufgenommen wurde, lässt sich daraus ablei-

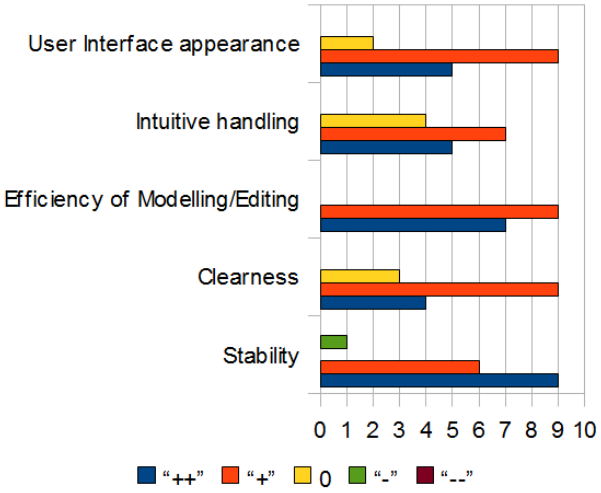


Abbildung 6.9.: Bewertung einzelner Anwendungskriterien

ten, dass alle Probanden angaben den Editor zur Erstellung dienstbasierter Anwendungen in der Praxis tatsächlich anwenden würden.

Als Hauptkritikpunkt muss die Umsetzung des Verknüpfungswerkzeugs zum Erstellen von Parameterabbildungen gesehen werden. Hier könnte eine Verbesserung der Benutzerfreundlichkeit des Werkzeugs durch die Optimierung der Abhängigkeit zwischen Navigations- und ParameterMapping-werkzeug erreicht werden.

6.3. Evaluierung des Gesamtkonzepts

Da das Resultat dieser Arbeit nur einen Teil der Prozesskette zur Generierung interaktiver, dienstbasierter Anwendungen darstellt, soll im Folgenden eine Evaluierung der gesamten Prozesskette erfolgen. Abbildung 6.10 gibt hierzu einen Überblick über die im Rahmen des Gesamtansatzes entwickelten Systemkomponenten.

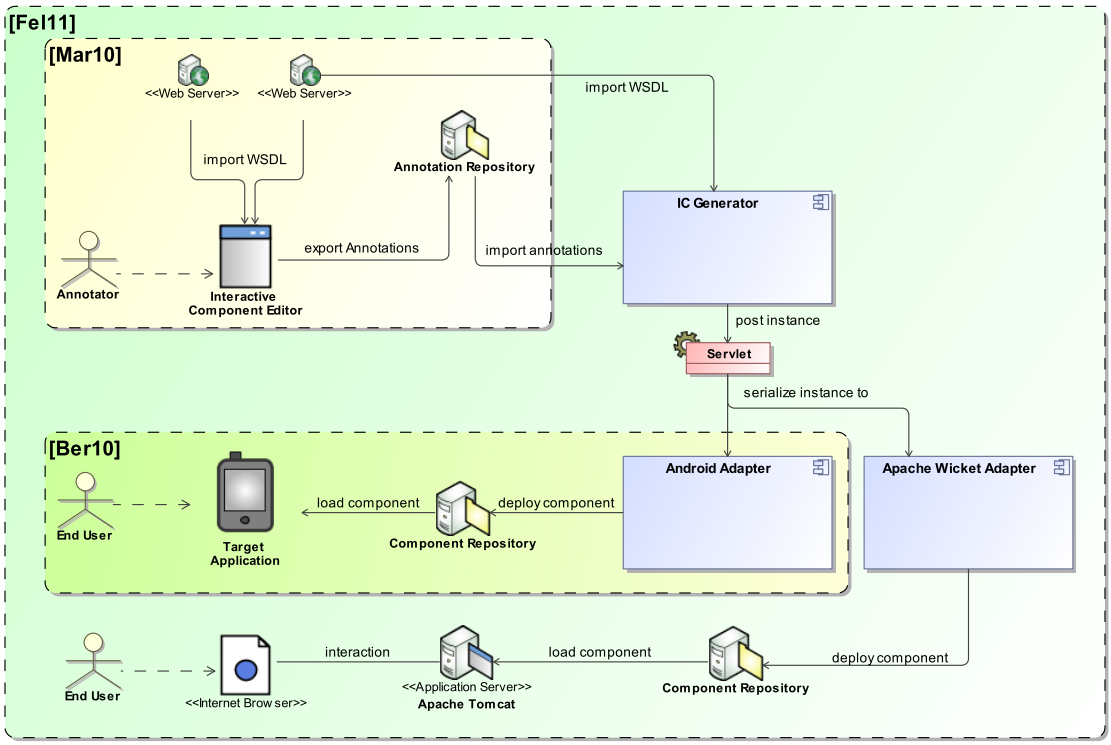


Abbildung 6.10.: Überblick über das Gesamtsystem

Der erste Schritt der Anwendungserstellung besteht in der Anwendungsbeschreibung. Hierzu wird von einem Anwendungsersteller der im Rahmen dieser Arbeit umgesetzte *Interactive Component Editor* eingesetzt. Der graphische Editor ermöglicht eine werkzeuggestützte Modellierung von dienstbasierten Anwendungen mit einer individuellen Navigationsstruktur und speziellen Benutzeroberflächenfunktionalitäten. Der Umfang der Anwendungs- beziehungsweise Benutzeroberflächenbeschreibung wird durch das in Abschnitt 4.1 konzipierte Annotationsmodell aufgespannt. Das Annotationsmodell dient ebenfalls dazu, die modellierten Anwendungen als Annotationsdateien in einem speziellen Repository zu veröffentlichen.

In der zweiten großen Phase des Gesamtansatzes, in Abbildung 6.10 mit [Fel11] gekennzeichnet, dient die Anwendungsbeschreibung des Annotation-Repositories dem *IC Generator* als Grundlage zur Generierung einer Instanz zur Beschreibung einer interaktiven Komponente in Form einer SBIC-Instanz. Hierbei werden mehrere M2M-Transformation durchgeführt, wobei die Paginierung und das Ableiten der konkreten Interaktoren für verschiedene Zielplattformen, nach speziell für die Plattform definierten Regeln erfolgt. Die

plattformspezifischen SBIC-Instanzen werden dann dem jeweiligen Plattformadapter zur Verfügung gestellt.

Die Plattformadapter überführen dann die jeweils plattformspezifische SBIC-Instanz in eine ausführbare interaktive Komponente, welche auf der jeweiligen Plattform integriert werden kann. Im Rahmen der Arbeit [Ber10] wurde ein konkreter Plattformadapter für *Android* realisiert. Neben diesem Plattformadapter wurde im Rahmen der Arbeit [Fel11] prototypisch ein weiterer Adapter umgesetzt, welcher interaktive Komponenten in Form von *Apache Wicket-Komponenten* generiert. Diese generierten Komponenten können in eine Zielanwendung integriert werden, welche dem Endnutzer als Web-Anwendung zu Verfügung gestellt wird.

Nachdem die Anwendungsmodellierung des in Absatz 3.1 aufgestellten Anwendungsszenarios mit dem Interactive Component Editor in Abschnitt 6.1 evaluiert wurde, zeigt Abbildung 6.11 die vom Android-Plattformadapter generierte Anwendung des Szenarios auf einem Android-Zielgerät.

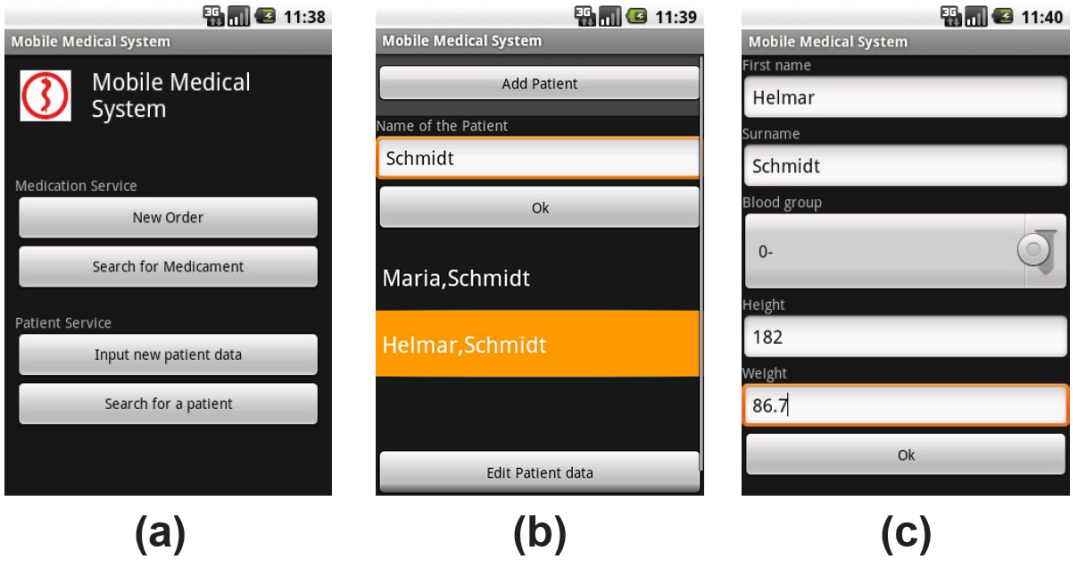


Abbildung 6.11.: Generierte Anwendung auf dem Android Zielgerät

Das Bildschirmfoto (a) zeigt die Startseite der generierten Anwendung. Diese verdeutlicht, dass für die Umsetzung der Anwendung die Dienste *Medication Service* und *Patient Service* integriert wurden und initial jeweils zwei Dienstoperationen zur Verfügung stellen. Bildschirmfoto (b) zeigt die Anwendungsseite für die Dienstoperation *searchPatient* des Patientendienstes, welche das Suchen nach Patienten ermöglicht. Hier sind neben dem Sucheingabefeld und der Suchergebnisliste, welche durch Anwendung der Dienstannotation *DisplayAtomicInOut* auf einer Seite dargestellt werden, auch die zwei modellierten Navigationsmöglichkeiten dieser Seite durch die Schaltflächen *Add Patient* und *Edit Patient Data* dargestellt. Nach der Selektion eines Patienten in der Suchergebnisliste und des Aufrufs der Navigation *Edit Patient Data* wird die Anwendungsseite zur Bearbeitung von Patientendaten dargestellt (Bildschirmfoto (c)). Hierzu dient wie im Anwendungsszenario

beschrieben die Dienstoperation *addPatient*. Die Daten des Patienten werden aufgrund der modellierten Parameterabbildungen automatisch in die Eingabefelder eingetragen.

Neben dem Anwendungsszenario dieser Arbeit wurde mit dem Editor auch das Anwendungsszenario aus [Ber10] vollständig modelliert und erfolgreich zu interaktiven Komponenten für Android und Wicket weiterverarbeitet.

6.4. Evaluierung der Anforderungen

Evaluierung der Anforderungen an das Softwarewerkzeug

/AS1: Benutzbarkeit

Der Editor ermöglicht durch die graphische Benutzeroberfläche und mit Hilfe unterschiedlicher Werkzeuge die Möglichkeit Dienstannotationen zu definieren. Wie in den vorherigen Absätzen demonstriert, lässt sich mit dem Softwarewerkzeug zielgerichtet eine interaktive, dienstbasierte Anwendung beschreiben. Um eine eindeutige Aussage zur Benutzbarkeit treffen zu können, wurde ein umfangreicher Usability-Test mit einer repräsentativen Gruppe von Nutzern durchgeführt. Die Testergebnisse dieser Auswertung wurden in Absatz 6.2 ausführlich analysiert.

/AS2: Erweiterbarkeit und Modularität

Der Editor wurde aufgrund seiner komponentenbasiert entworfenen Systemarchitektur modular umgesetzt und erfüllt dadurch die Eigenschaft der Erweiterbarkeit und Änderbarkeit. Die separat entwickelten Hauptkomponenten wie beispielsweise der WSDL-Parser oder das Servicemodell können unabhängig von der Systemarchitektur für beliebig andere Systeme eingesetzt werden. Ebenfalls können einzelne Komponenten leicht erweitert beziehungsweise komplett ersetzt werden. Zusätzlich unterstützt wurde die modulare Entwicklung durch die Verwendung der Plug-in-Entwicklungsumgebung von Eclipse.

/AS3: Serialisierung einer erstellten Anwendung in ein festgelegtes Ziel-Modell

Der Editor verfügt über eine Exportfunktion, welche alle modellierten Dienstannotationen einer erstellten Anwendung als Instanz des in Abschnitt 4.1 vorgestellten erweiterten ServFace-Annotationmodels serialisiert. Hierzu wird sowohl das lokale Abspeichern, als auch die Anbindung an einen speziellen Verzeichnisdienst für eine verbesserte Verwaltung von verfügbaren Anwendungen, unterstützt. Die mit dem Editor serialisierten Dienstannotationen können über eine Importfunktion wieder Deserialisiert werden, wodurch der Anwendungszustand vollständig wiederhergestellt und eine Weiterbearbeitung ermöglicht wird. Listing A.2 zeigt als Beispiel die serialisierte Instanz einer Anwendung anhand des Anwendungsszenarios.

Evaluierung der Anforderungen an die Dienstannotationen

/AD1: Beschreibung von Navigations- und Datenflüssen

Die Erstellung von Navigations- und Datenflüssen wurde innerhalb des Navigation-Editors durch zwei separate Verknüpfungswerkzeuge umgesetzt, so dass der Nutzer die Modellierung von Zustandsübergängen und Parameterabbildungen möglichst intuitiv anwenden kann. Navigations- und Datenflüsse werden dabei durch die konzipierte Dienstannotation *NavigationChoice* beschrieben und serialisiert. Zusätzlich erhält der Nutzer mit Hilfe des Overview-Editors eine Übersicht über alle bereits erstellten Navigations- und Datenflüsse.

/AD2: Beschreibung erweiterter Funktionalitäten, die mit ServFace-Annotationen nicht ausgedrückt werden können

Zur Beschreibung erweiterter Funktionalitäten, die wiederkehrende Eigenschaften von Benutzerschnittstellenelementen in Anwendungen ausdrücken, wurden die in Kapitel 4.1 vorgestellten Dienstannotationen entworfen. Alle aufgeführten Dienstannotationen lassen sich mit Hilfe des graphischen Editors formulieren. Darüber hinaus wurde eine Schnittstelle geschaffen, um zusätzlich Dienstannotationen die im Rahmen des ServFace-Projektes entstanden sind, zu beschreiben.

/AD3: Unterstützung verschiedener Zielplattformen

Das für die Umsetzung der Dienstannotationen herangezogene ServFace-Annotationsmodell unterstützt die Bindung von Dienstannotationen an Plattformspezifikationen. Da diese Eigenschaft bei der Konzeption der Dienstannotationen genutzt wurde, ist eine Beschreibung unterschiedlicher Dienstannotationen für verschiedene Plattformen somit gewährleistet und wird auch vom Editor unterstützt.

/AD4: Unterstützung erweiterter Kontextbeschreibungen

Die entwickelten Dienstannotationen wurden hinsichtlich geeigneter Bindungen an Kontextinformationen analysiert und gegebenenfalls um Kontextbeschreibungen erweitert. Hierzu wurde der Kontextbeschreibungsmechanismus des ServFace-Annotationmodels aufgegriffen und erweitert, so dass neben den Plattformrestriktionen auch Einschränkungen bestimmter Dienstannotationen gegenüber Sprachen und Standorten ausgedrückt werden können.

Übergeordnete Anforderung

AÜ1: Integration in den Gesamtansatz

Wie in Abschnitt 6.3 demonstriert, wurde das entwickelte Beschreibungsformat der Dienstannotationen und der graphische Editor erfolgreich in das Gesamtsystem integriert. Abbildung 6.11 zeigt, dass die automatische Generierung einer mit dem Editor modellierten Anwendung erfolgreich durchgeführt werden kann.

7. Zusammenfassung und Ausblick

Das Ziel der vorliegenden Arbeit war es, einen graphischen Editor zur Modellierung von interaktiven, dienstbasierten Anwendungen zu erstellen. Die modellierten Anwendungen sollten nur mit Hilfe von Dienstannotationen ausgedrückt werden können und insbesondere eine Beschreibung von Navigations- und Datenflüssen ermöglichen. Die Motivation des Ansatzes bestand darin, Anwendungen auf Basis von Dienstfunktionalitäten ohne Programmieraufwand beschreiben und auf unterschiedlichen Plattformen zur Ausführung integrieren zu können. Diese Arbeit stellt hierbei den ersten Teil der Prozesskette des Gesamtkonzeptes dar. In den weiteren Prozessschritten werden die vom Editor erzeugten Anwendungsbeschreibungen herangezogen, um nach Anwendung mehrerer Transformationsschritte plattformspezifische Anwendungen zu generieren und in eine Zielplattform zu integrieren (vgl. Abschnitt 6.3).

In einem ersten Schritt wurden die Grundlagen zur Konzeption der Dienstannotationen und Umsetzung des Softwarewerkzeugs gelegt. Ausgehend von den allgemeinen Techniken zur automatisierten Generierung von Benutzerschnittstellen für Dienste, wurden in einer Untersuchung des aktuellen Stands der Technik zunächst bestehende Annotationsformate zur Beschreibung von Benutzerschnittstellen für Dienste untersucht und bewertet. Dabei hat sich herausgestellt, dass das im Rahmen des ServFace-Projektes entstandene ServFace-Annotationsmodell eine geeignete Grundlage zur Beschreibung der Dienstannotationen bietet. Im zweiten Teil der Untersuchung zum aktuellen Stand der Technik wurden sowohl bestehende Editoren zum Erstellen von Dienstannotationen als auch Editoren zur Beschreibung interaktiver, dienstbasierter Anwendungen untersucht. Hierbei konnte herausgestellt werden, dass derzeit kein Editor existiert, der beide Eigenschaften erfüllt. Nach der Feststellung der Notwendigkeit des neuen Ansatzes, wurde im Kapitel 3 ein geeignetes Anwendungsszenario zur Identifikation der Anforderungen beschrieben. Basierend auf den Grundlagen und den formulierten Anforderungen stellt Kapitel 4 das Konzept dieser Arbeit vor. Es wurden geeignete Dienstannotationen zur Beschreibung von interaktiven, dienstbasierten Anwendungen und ein Systemarchitektur zur Umsetzung eines graphischen Editors vorgestellt. Das entworfene Konzept wurde umgesetzt und in Kapitel 6 anhand des beschriebenen Anwendungsszenarios hinsichtlich der Funktionalität erfolgreich validiert. Nach einer ebenfalls erfolgreichen Evaluierung der aufgestellten Anforderungen, wurde die Umsetzung des Gesamtkonzeptes von der Anwendungsmodellierung bis zur Ausführung der Anwendung auf dem Zielgerät vorgestellt und bewertet.

Die in dieser Arbeit entstandenen Ergebnisse zeigen, dass mit dem Editor interaktive, dienstbasierte Anwendungen modelliert und vollständig als Dienstannotationen beschrieben werden können. Die Ergebnisse der weiteren Arbeiten des Gesamtkonzeptes zeigen

auch, dass aus den Anwendungsbeschreibungen lauffähige Anwendungen generiert werden können. Derzeit wird die Generierung für die Android Plattform und für das Apache Wicket Framework unterstützt. Eine Unterstützung weiterer Plattformen ist für die Zukunft denkbar.

Für den prototypisch umgesetzten Editor sind folgende Erweiterungen denkbar:

- Aus der Auswertung des Usability-Tests konnten Verbesserungsmöglichkeiten für den Editor abgeleitet werden. Vor allem eine Optimierung der Werkzeuge zum Erstellen von Parameterabbildungen würde sich positiv auf die Benutzerfreundlichkeit auswirken.
- Im umgesetzten Prototyp des Editors wurde bereits ein Dialog zur Definition von globalen Layouteigenschaften für Anwendungen eingeführt. Um dem IC-Generator die Layouteigenschaften für die Anwendungsgenerierung zur Verfügung stellen zu können, muss ein geeignetes Beschreibungsformat für die Layouteigenschaften entwickelt und eine Exportfunktion in den Editor integriert werden.
- Da der Prototyp die Anwendung von nur vier ServFace-Dienstannotationen unterstützt, könnte der Beschreibungsumfang für Anwendungen erhöht werden, indem weitere Dienstannotationen die im Rahmen des ServFace Projektes entstanden sind, unterstützt werden. Weiterhin lassen sich in weiteren Anwendungsszenarios neue Anwendungsfunktionalitäten finden, die sich mit den bestehenden Dienstannotationen nicht beschreiben lassen. Hier würde eine Erweiterung des Annotationenmodells und die Integration in den Editor den Beschreibungsumfang zusätzlich erhöhen.
- Der Editor kann um weitere Editoransichten erweitert werden. So könnte beispielsweise eine Editoransicht integriert werden, um datenbankbasierte Dienste mit einfachen Datenmanipulationsfunktionalitäten zu erstellen, um diese Dienste in die anschließende Anwendungsmodellierung einbeziehen zu können. Dieser erweiterte Ansatz zur dienstbasierten Anwendungsmodellierung, welcher in einem ersten Schritt die Generierung von benutzerdefinierten Diensten vorsieht, wird in [FOG+10] beschrieben.

A. Anhang

A.1. Aufgabenstellungen und Fragebogen des Usability-Tests

Evaluation – Group 1

Task

The goal is to create a service-based medical application, for patient and medicament management, by use of the *Interactive Component Editor.* For this purpose two services provide the required functionalities. The service descriptions of these services are located at the given URIs.

- **Patient-Service:**
  http://141.76.40.51:8080/Patient_Service/services/PatientService?wsdl

- **Medicament-Service:**
  http://141.76.40.51:8080/Medicament_Service/services/MedicamentService?wsdl

Firstly, a new application with the name „Medical Application“ and a relation to the ***Patient-Service*** has to be created. In order to use the functionalities of the ***MedicamentService*** as well, this service has to be imported additionaly in a second step.

After that, add pages for the service-operations ***addPatient, searchPatient, searchMedicament*** and ***newOrder.*** Of these the operations ***searchPatient*** and ***searchMedicament*** need to be marked as initial operations.

The next step is to create a navigation-path from the page ***searchPatient*** to the page ***addPatient.*** For this purpose a ***Navigation-Choice*** with one option and a suitable label need to be created at the source-page. The navigation-path can then be connected between the option of the Navigation-Choice and the related operation of the target-page.

In the same way, create a further navigation-path from the page ***searchMedicament*** to the page ***newOrder***. Additionally attach a Parameter-Mapping from the return-parameter ***pzn*** of ***searchMedicament*** to the input-parameter ***pzn*** of the page ***newOrder.*** Check the created navigation-paths and parameter-mapping with the OverviewEditor.

At last, customize the display of the complex return-parameter ***return*** of the page ***searchMedicament*** by editing a HumanReadableString. Choose the subelements ***name***, ***producer*** and ***price***.

Finish by exporting the application!

Evaluation – Group 2

For the creation of a service-based application two services are provide at the given URIs.

- **Patient-Service:**

http://141.76.40.51:8080/Patient_Service/services/PatientService?wsdl

- **Medicament-Service:**
  http://141.76.40.51:8080/Medicament_Service/services/MedicamentService?wsdl

The **Patient-Service** enables the management of patient data for a doctor's office by the following relevant service-operations:

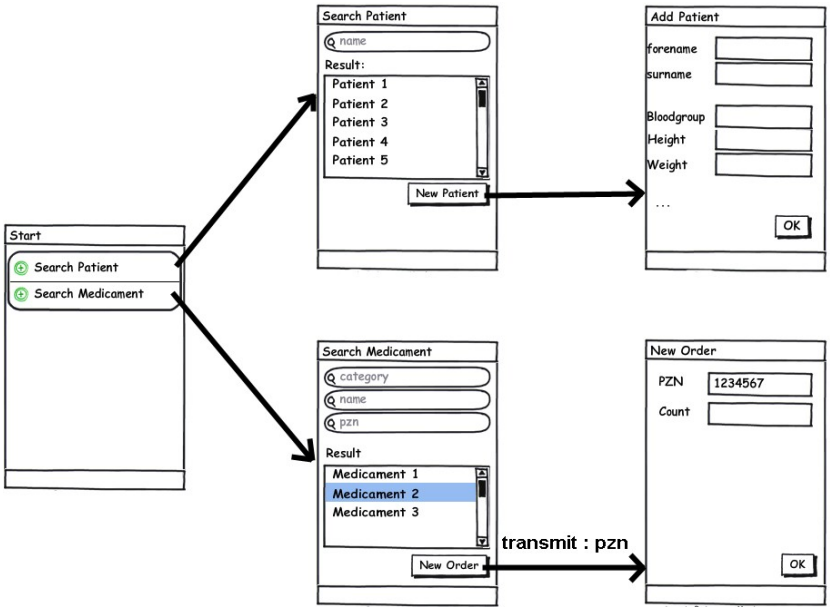
- **addPatient** → adds a new Patient to the patient-database
- **searchPatient** → search for patients in the database by the patient-name

The **Medicament-Service** offers functionalities for the medicament-management by the following relevant service-operations:

- **getOrders** → get a List of all previously ordered medicaments
- **newOrder** → create a new order of a medicament
- **searchMedicament** → search for a medicament in the medicament-database
- **showMedicamentDetails** → get detailed information about a medicament

Task

Your task is to create a medical application for patient and medicament management, by use of the *Interactive Component Editor*. The whole application-structure with the resulting user interfaces and Navigation-Paths is illustrated by the following figure.



The resulting application should offer the functionalities to **search for patients** and to **search for medicaments** as initial operations at the startpage.
From the page that provides the user interface to **search for patients**, the user should have the posibility to navigate to the page for **adding a new patient**.
From the page that provides the user interface to **search for medicaments**, the user should be able to navigate to the page for a **new order** of medicaments. For this navigation the parameter **pzn** has to be transmited to the corresponding input field of the target-page.
Finish by exporting the annotations.

Questionary

Was the structure of the Interactive Component Editor clear and comprehensible?

[ ] very clear and comprehensible(++)
[ ] clear and comprehensible(+)
[ ] indifferent(0)
[ ] not really clear and comprehensible(-)
[ ] absolutely not clear and comprehensible(--)

Did you find all editor-functions and tools to accomplish the given task?

[ ] yes
[ ] no, (which tool or function could not be found?):

_____

_____

_____

Did any functional problems occur by utilizing the tools?

[ ] no
[ ] yes, (what kind of problem with which tool?):

_____

_____

_____

How do you rate the significance of used labels and icons in the Interactive Component Editor?

[ ] very comprehensible(++)
[ ] comprehensible(+)
[ ] indifferent(0)
[ ] not really comprehensible(-)
[ ] absolutely not comprehensible(--)

How useful do you rate the Overview-Editor?

[ ] very useful(++)
[ ] useful(+)
[ ] indifferent(0)
[ ] not really useful(-)
[ ] absolutely not useful(--)

How comfortably is the creation of Navigation-Paths with the Navigation Path-connection-tool?

[ ] very comfortably(++)
[ ] comfortably(+)
[ ] indifferent(0)
[ ] not really comfortable(-)
[ ] absolutely not comfortable

How comfortably is the creation of Parameter-Mappings with the Parameter Mapping-connection-tool?

[ ] very comfortably(++)
[ ] comfortably(+)
[ ] indifferent(0)
[ ] not really comfortable(-)
[ ] absolutely not comfortable

Would you use the Interactive Component Editor for the creation of service-based applications instead of manually developing service-based applications?

[ ] yes
[ ] no

Score the Interactive Component Editor by the following criteria:

Criteria	**Very good(++)**	**Good(+)**	**Satisfying(0)**	**Insufficient(-)**	**bad(--)**
Stability					
Clearness					
Efficiency of Modelling/Editing					
Intuitive handling					
User Interface appearance					

Place for further comments or hints:

_____

_____

_____

_____

_____

Thank you for participating!

A.2. Dienstannotationen der Patientenverwaltung

```
<?xml version="1.0" encoding="ASCII"?>
<servfaceannotationmodel:AnnotationModel xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:annotations="http://annotations/1.2
    " xmlns:deltaAnnotations="http://deltaAnnotations" xmlns:servfaceannotationmodel="http://
    servfaceannotationmodel/1.2" baseDescriptionFileURI="" baseNamespace="" root="//
    @referenceObjects.0">
  <referenceObjects hierarchicalName="PatientService" descriptionFileURI="http://141.76.40.51:8080/
      Patient_Service/services/PatientService?wsdl">
    <annotations xsi:type="deltaAnnotations:Application" name="Medical Application"
        initialOperations="//@referenceObjects.1 //@referenceObjects.2" treatedOperations="//
        @referenceObjects.1 //@referenceObjects.2"/>
  </referenceObjects>
  <referenceObjects hierarchicalName="PatientService.searchPatient" descriptionFileURI="http:
      //141.76.40.51:8080/Patient_Service/services/PatientService?wsdl" refType="Operation">
    <annotations xsi:type="deltaAnnotations:DisplayAtomicInOut"/>
  <annotations xsi:type="annotations:TextFeedback" text="Add new Patient"/>
    <annotations xsi:type="annotations:TextFeedback" text="Edit Patient data"/>
    <annotations xsi:type="deltaAnnotations:NavigationChoice">
      <options target="//@referenceObjects.2" defaultOptionLabel="//@referenceObjects.1/@annotations
          .0">
        <transmits type="NONE"/>
      </options>
      <options target="//@referenceObjects.2" defaultOptionLabel="//@referenceObjects.1/@annotations
          .1">
        <transmits>
          <parameterMappings source="//@referenceObjects.8" target="//@referenceObjects.10"/>
          <parameterMappings source="//@referenceObjects.3" target="//@referenceObjects.11"/>
          <parameterMappings source="//@referenceObjects.4" target="//@referenceObjects.12"/>
          <parameterMappings source="//@referenceObjects.6" target="//@referenceObjects.13"/>
          <parameterMappings source="//@referenceObjects.7" target="//@referenceObjects.14"/>
          <parameterMappings source="//@referenceObjects.9" target="//@referenceObjects.15"/>
        </transmits>
      </options>
    </annotations>
  </referenceObjects>
  <referenceObjects hierarchicalName="PatientService.addPatient" descriptionFileURI="http:
      //141.76.40.51:8080/Patient_Service/services/PatientService?wsdl" refType="Operation"/>
  <referenceObjects hierarchicalName="PatientService.searchPatient.output.return.foreName"
      descriptionFileURI="http://141.76.40.51:8080/Patient_Service/services/PatientService?wsdl"
      refType="ParameterElement"/>
  <referenceObjects hierarchicalName="PatientService.searchPatient.output.return.surName"
      descriptionFileURI="http://141.76.40.51:8080/Patient_Service/services/PatientService?wsdl"
      refType="ParameterElement"/>
  <referenceObjects hierarchicalName="PatientService.searchPatient.output.return" descriptionFileURI
      ="http://141.76.40.51:8080/Patient_Service/services/PatientService?wsdl" refType="Parameter">
    <annotations xsi:type="deltaAnnotations:HumanReadableString" parameterElements="//
        @referenceObjects.3 //@referenceObjects.4" separationChar=","/>
  </referenceObjects>
  <referenceObjects hierarchicalName="PatientService.searchPatient.output.return.bloodGroup"
      descriptionFileURI="http://141.76.40.51:8080/Patient_Service/services/PatientService?wsdl"
      refType="ParameterElement"/>
  <referenceObjects hierarchicalName="PatientService.searchPatient.output.return.height"
      descriptionFileURI="http://141.76.40.51:8080/Patient_Service/services/PatientService?wsdl"
      refType="ParameterElement"/>
  <referenceObjects hierarchicalName="PatientService.searchPatient.output.return.id"
      descriptionFileURI="http://141.76.40.51:8080/Patient_Service/services/PatientService?wsdl"
      refType="ParameterElement"/>
  <referenceObjects hierarchicalName="PatientService.searchPatient.output.return.weight"
      descriptionFileURI="http://141.76.40.51:8080/Patient_Service/services/PatientService?wsdl"
      refType="ParameterElement"/>
  <referenceObjects hierarchicalName="PatientService.addPatient.input.id" descriptionFileURI="http:
      //141.76.40.51:8080/Patient_Service/services/PatientService?wsdl" refType="Parameter">
    <annotations xsi:type="deltaAnnotations:DisabledParameter"/>
  </referenceObjects>
  <referenceObjects hierarchicalName="PatientService.addPatient.input.foreName" descriptionFileURI="
      http://141.76.40.51:8080/Patient_Service/services/PatientService?wsdl" refType="Parameter"/>
  <referenceObjects hierarchicalName="PatientService.addPatient.input.surName" descriptionFileURI="
      http://141.76.40.51:8080/Patient_Service/services/PatientService?wsdl" refType="Parameter"/>
```

```
41  <referenceObjects hierarchicalName="PatientService.addPatient.input.bloodGroup" descriptionFileURI
        ="http://141.76.40.51:8080/Patient_Service/services/PatientService?wsdl" refType="Parameter"/
        >
42  <referenceObjects hierarchicalName="PatientService.addPatient.input.height" descriptionFileURI="
        http://141.76.40.51:8080/Patient_Service/services/PatientService?wsdl" refType="Parameter"/>
43  <referenceObjects hierarchicalName="PatientService.addPatient.input.weight" descriptionFileURI="
        http://141.76.40.51:8080/Patient_Service/services/PatientService?wsdl" refType="Parameter"/>
44  <referenceObjects hierarchicalName="PatientService.addPatient.input.latitude" descriptionFileURI="
        http://141.76.40.51:8080/Patient_Service/services/PatientService?wsdl" refType="Parameter">
45    <annotations xsi:type="deltaAnnotations:PlatformProvidedEntity" platforms="//@platforms.0"
          locations="//@locations.0">
46      <invocationDetails qualifiedName="android.gps.services.GPSChecker" methodName="getLatitude">
47        <componentURI>http://download.btown.de/components/AndroidGPSServices.jar</componentURI>
48        <componentURI>http://download.btown.de/components/AndroidGPSChecker.jar</componentURI>
49      </invocationDetails>
50    </annotations>
51  </referenceObjects>
52  <referenceObjects hierarchicalName="PatientService.addPatient.input.longitude" descriptionFileURI=
        "http://141.76.40.51:8080/Patient_Service/services/PatientService?wsdl" refType="Parameter">
53    <annotations xsi:type="deltaAnnotations:PlatformProvidedEntity" platforms="//@platforms.0"
          locations="//@locations.0">
54      <invocationDetails qualifiedName="android.gps.services.GPSChecker" methodName="getLongitude">
55        <componentURI>http://download.btown.de/components/AndroidGPSServices.jar</componentURI>
56        <componentURI>http://download.btown.de/components/AndroidGPSChecker.jar</componentURI>
57      </invocationDetails>
58    </annotations>
59  </referenceObjects>
60  <platforms id="android"/>
61  <locations id="home"/>
62 </servfaceannotationmodel:AnnotationModel>
```

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Literaturverzeichnis

[Ber09] Georg Berndt. Ad-hoc integration of annotated services to instances of a meta-model of interactive service based applications. Großer beleg, Dresden University of Technologie, 2009. 5.1.1

[Ber10] Georg Berndt. Generierung von interaktiven OSGi-Komponenten für Google-Android. Diplomarbeit, Dresden University of Technologie, 2010. 1.2, 4.1.12, 6.3, 6.3

[Bur04] Lassila Mcdermott Mcilraith Narayanan Paolucci Parsia Payne Sirin Srinivasan Sycara Katia Burstein, Hobbs. Owl-s: Semantic markup for web services. Website, November 2004. 2.3.1.4

[Chr01] Meredith Weerawarana Sanjiva Christensen, Curbera. Web service definition language (wsdl). Technical report, March 2001. 4.2.3, 5.1

[CTT] Concurrent task tree environment, see http://giove.cnuce.cnr.it/ctte.html, last visited on 2007-08-10. 1

[Fel11] Marius Feldmann. Ein Verfahren zur Generierung interaktiver Komponenten aus annotierten funktionalen Dienstschnittstellenbeschreibungen. Dissertation, Dresden University of Technologie, 2011. 1.2, 1.2, 3.1, 3.2, 4.1.7, 4.1.12, 4.2.5, 5.1.1, 6.3

[FHLS10] Marius Feldmann, Gerald Hübsch, Christian Liebing, and Alexander Schill. Anwendung dienstorientierter Ansätze im Kontext der Erstellung von Benutzungsschnittstellen für serviceorientierte Applikationen. *PIK - Praxis der Informationsverarbeitung und Kommunikation*, (Band 33, Heft 2):147 – 153, 2010. 4.1.8

[FNM+09] Marius Feldmann, Nestler, Muthmann, Jugel, Hübsch, and Alexander Schill. Overview of an end-user enabled model-driven development approach for interactive applications based on annotated services. In *WEWST '09: Proceedings of the 4th Workshop on Emerging Web Services Technology*, pages 19–28, New York, NY, USA, 2009. ACM.

[FOG+10] Marius Feldmann, Vladyslav Oleniuk, Larissa Globa, Alexander, and Schill. Rapid development of data-intensive web services. Technical report, 2010. 7

[Gar09] Gartner. Gartner's hype cycle special report for 2009. Technical report, Gartner Research, 2009. 1.1

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995. 4.1, 5.1.1

[IJH+09] Paoli Izquierdo, Jordan Janeiro, Gerald Hübsch, Thomas, Springer, and Alexander Schill. An annotation tool for enhancing the user interface generation process for services. *Microwave and Telecommunication Technology*, 2009. 2.3.2.2

[Jug10] Uwe Jugel. Service annotations for user interface composition - methodology for service-based interactive application development. Technical report, Project ServFace, 2010. 2.3.2.2

[Kaw04] Jalal Kawash. Declarative user interfaces for handheld devices. In *WISICT '04: Proceedings of the winter international synposium on Information and communication technologies*, pages 1–6. Trinity College Dublin, 2004. 2.3.1.3

[KKM03] Michael Kassoff, Daishi Kato, and Waqar Mohsin. Creating GUIs for Web Services. *IEEE Internet Computing*, 7(5):66–73, 2003. 2.3.1.1

[KL05] Deepali Khushraj and Ora Lassila. Ontological approach to generating personalized user interfaces for web services. pages 916–927. 2005. 2.3.1.4

[LMS10] Christian Liebing, Ronny Mennerich, and Alexander Schill. A pragmatic approach for matching UI components on Web Service operations. Miami, 2010. 4.1.8

[Mar09] Felix Martens. Ad-hoc Integration von annotierten Diensten in Google-Android-Applikationen. Großer beleg, Dresden University of Technologie, 2009. 2.3.2.2, 5.1.1

[ML05] Jeff McAffer and Jean-Michel Lemieux. *Eclipse Rich Client Platform: Designing, Coding, and Packaging Java(TM) Applications*. Addison-Wesley Professional, 2005.

[MVA10] Jeff McAffer, Paul VanderLei, and Simon Archer. *OSGi and Equinox: Creating Highly Modular Java Systems*. Addison-Wesley Professional, 2010.

[NFPS09] Tobias Nestler, Marius Feldmann, Andre Preußner, and Alexander Schill. Service composition at the presentation layer using web service annotations. CEUR workshop proceedings, 2009.

[Pro09] Servface Project. Servface project. 2009. 2.3.1.2

[PSSD09] Paterno', Fabio Santoro, Carmen Spano, and Lucio Davide. Maria: A universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments. *ACM Trans. Comput.-Hum. Interact.*, 16(4):1–30, 2009. 2.3.2.4

[SBPM08] Dave Steinberg, Frank Budinsky, Marcelo Paternostro, and Ed Merks. *EMF: Eclipse Modeling Framework (2nd Edition)*. Addison-Wesley Professional, 2008.

[Spi06a] J. Spillner. Project dynvocation. Master's thesis, TU-Dresden, 2006. 1.1, 2.3.1.1

[Spi06b] Josef Spillner. Entwicklung eines Editors zum Entwurf von Benutzerschnittstellen für Web Services auf Basis der abstrakten UI-Beschreibungssprache WSGUI. Master's thesis, TU-Dresden, 2006. 2.3.2.1

[W3C07] W3C. Web services description language (wsdl) version 2.0 part 1: Core language, Mai 2007.